DYEGO ROGHER DREES

VISUALIZAÇÃO 3D INTERATIVA UTILIZANDO UM SISTEMA DE RENDERIZAÇÃO REMOTA

CURITIBA

2010

DYEGO ROGHER DREES

VISUALIZAÇÃO 3D INTERATIVA UTILIZANDO UM SISTEMA DE RENDERIZAÇÃO REMOTA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luciano Silva
Co-Orientadora: Profa. Dra. Olga Regina Pereira Bellon

CURITIBA

2010

AGRADECIMENTOS

Aos professores Olga Regina Pereira Bellon e Luciano Silva, pela orientação e amizade. Agradeço a oportunidade de participar do Grupo de Pesquisa IMAGO.

Ao CNPq, FINEP, Fundação Araucária e CAPES pelo financiamento deste mestrado, a Caroline Mazetto Mendes por todos os momentos de apoio e a todos os colegas do IMAGO.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

RESUMO

O desenvolvimento de novas tecnologias de imageamento e modelagem 3D de objetos com precisão para preservação digital de acervos tem ganhado atenção da comunidade científica nos últimos anos. Nesse contexto, a disponibilização e a segurança durante o acesso desses acervos digitais são temas de grande importância. Acessar os conteúdos disponibilizados por museus virtuais por exemplo, a partir de um simples computador conectado à Internet e visualizar modelos 3D com realismo ainda é um desafio.

Com a evolução dos equipamentos de aquisição de dados 3D, objetos estão sendo cada vez mais fielmente reproduzidos, resultando em modelos 3D realistas que podem possuir direitos autorais. Deve-se assim, permitir a visualização de detalhes de modelos 3D preservando os direitos autorais das obras. Através da renderização remota pode-se garantir a segurança dos modelos 3D, que ficam armazenados em um servidor, e ao mesmo tempo usuários podem visualizar detalhes dos modelos 3D para realização de pesquisas e atividades educacionais.

Este trabalho apresenta um estudo sobre técnicas de visualização de modelos 3D utilizando renderização remota, destacando suas vantagens e desvantagens com relação à segurança e interatividade do usuário com o conteúdo 3D. São apresentados também quais métodos de acesso a modelos 3D são utilizados pelos principais projetos nesta área, mostrando a implementação do sistema de renderização remota desenvolvido que permite acesso ao conteúdo 3D disponibilizado de forma rápida, segura e que não necessite de *hardware* especial para tal.

ABSTRACT

The development of new technologies for precise imaging and 3D modeling of objects for digital preservation of assets has gained the scientific community's attention in last years. In this context, the disponibilization and safety during access to these digital collections is a constant concern. Accessing the content available by virtual museums for example, from a simple computer connected to the Internet and view 3D models with realism is still a challenge.

With the evolution of acquisition equipments of 3D data, objects are more accurately reproduced, resulting in realistic 3D models that can have copyright. It should therefore allow the visualization of details of 3D models preserving their copyrighted. Through remote rendering can guarantee the security of 3D models, that are stored on a server, while users can view details of the 3D models for conducting research and educational activities.

This work presents a study of techniques for visualization of 3D models using remote rendering, highlighting their advantages and disadvantages on the security and user interactivity with the 3D content. It also shows which methods of access to 3D models are used by major projects in this area, showing the implementation of the rendering system that enables remote access to 3D models without special hardware and in a safe and fast way.

CAPÍTULO 1

INTRODUÇÃO

A preservação da informação é muito importante para o desenvolvimento da nossa civilização [21]. Manter as informações em meios digitais permite às gerações futuras ter maior conhecimento sobre sua história e cultura. Artefatos e obras de arte pertencentes aos museus podem ser preservados digitalmente, auxiliando as atividades de pesquisa através da visualização remota.

Os museus virtuais apresentam uma nova forma de acesso ao conhecimento na qual pessoas podem visitar, no momento em que desejarem, qualquer museu do mundo que esteja disponível na Internet. Com o advento da realidade virtual, museus que possuem obras de arte, como esculturas, podem disponibilizá-las de forma interativa na qual pessoas podem visualizar detalhes e ângulos variados das obras.

Uma das formas de representação digital de objetos é a utilização de modelos 3D, que permitem armazenar informações detalhadas sobre a geometria e textura dos mesmos. Com o passar dos anos, cada vez mais os modelos 3D estão sendo reproduzidos com maior fidelidade, graças aos avanços das tecnologias de aquisição, e consequentemente, mais dados são gerados. Modelos 3D valiosos, obtidos com a digitalização de alta resolução de patrimônios culturais, podem exigir proteção para evitar a pirataria ou utilização indevida [19]. Ao mesmo tempo, deve-se permitir a visualização interativa desses modelos 3D para um grande número de pessoas.

Assim como na Preservação Digital, outras áreas de aplicação também podem exigir segurança ao conteúdo 3D disponibilizado. Modelos 3D de objetos usados para divulgação e vendas na Internet e personagens 3D desenvolvidos para jogos e filmes requerem soluções para preservar os direitos autorais [19]. Similar à questão de segurança dos modelos 3D,

tornar os dados 3D acessíveis para fins de pesquisa e estudo remoto é algo que deve ser considerado. É necessário garantir que qualquer pessoa que possua um computador, mesmo sem placa aceleradora 3D, com acesso à Internet possa usufruir do conteúdo 3D disponibilizado.

Essas questões conduzem a uma interessante vertente na visualização remota. Possibilitar o acesso a grande quantidade de modelos 3D de objetos pode facilitar a divulgação de objetos culturais na Internet. Garantir boa qualidade na visualização e disponibilizar meios diferenciados de acesso ao conteúdo 3D são primordiais para auxiliar as atividades educacionais.

Entre as soluções existentes para visualização de conteúdo 3D na Internet, pode-se destacar a renderização remota [4]. Uma das estratégias mais usadas é utilizar um servidor para renderizar o modelo 3D e transmitir uma imagem, um *stream* ou um vídeo de curta duração para o cliente [10, 11, 19]. Gráficos remotos são elementos-chave para a visualização à distância [14].

Este trabalho está organizado conforme descrito a seguir. O Capítulo 2 introduz os desafios e cenários existentes em relação à renderização remota, apresentando projetos que utilizam esta técnica. Detalhes sobre o sistema de renderização remota desenvolvido e seus meios alternativos para permitir a visualização são apresentados no Capítulo 3. No Capítulo 4 é apresentado o Museu Virtual 3D do IMAGO, no qual o sistema foi incorporado. Por fim, no Capítulo 5 são expostas as conclusões e trabalhos futuros.

CAPÍTULO 2

ESTADO DA ARTE

A renderização em tempo real está preocupada em gerar rapidamente imagens no computador, sendo o tipo de renderização mais interativo na área de computação gráfica. Uma imagem aparece na tela, o usuário age ou reage a essa imagem, afetando a próxima resposta [1]. A renderização remota permite que dados 3D sejam renderizados em um servidor de acordo com ações do cliente, que deve receber as imagens da renderização o mais breve possível para a visualização.

Neste capítulo são apresentados os desafios de trabalhar com grande quantidade de dados 3D e os cenários existentes na visualização de dados 3D, considerando tanto o cliente quanto o servidor envolvidos. Algumas características que um servidor deve conter, principalmente com relação à segurança, também são apresentadas.

2.1 Desafios

Trabalhar com grande quantidade de dados 3D que deve ser visualizado através da Internet envolve trabalhar com diferentes tecnologias, além de exigir *hardware* especial. A seguir estão listados alguns desafios observados em [34, 39]:

- **Enorme quantidade de dados 3D**: As tecnologias de aquisição, como por exemplo *scanners* a *laser*, podem adquirir dados com incrível nível de detalhes, gerando grandes modelos 3D. As aplicações precisam se preocupar com o armazenamento, transmissão e visualização desses dados.

- ***Hardware* Especial:** A renderização de dados 3D envolve cálculos matemáticos, como por exemplo simulação de iluminação. Esses cálculos podem exigir *hardware* especial, como uma placa de vídeo com aceleração 3D, grande quantidade de memória e um processador potente.

- **Taxa de Quadros:** Para ser possível observar um movimento contínuo e suave, uma boa taxa de quadros por segundo é necessária. Em uma aplicação em tempo real, a taxa de pelo menos 15 a 20 quadros por segundo é exigida para haver interatividade [1, 6]. A taxa de quadros também deve ser constante e invariante a qual parte do conteúdo 3D esteja sendo observada, caso contrário, influenciará negativamente na interação do usuário com o conteúdo 3D.

- **Latência:** É o tempo medido a partir do momento de uma entrada até a manifestação da saída correspondente. Muitos fatores contribuem para latência: dispositivos de entrada, arquitetura do *software* utilizado, tempo de renderização, visualização final, entre outros.

- **Redes:** A Internet possui diferentes tipos de banda e estabilidade. É inevitável uma certa taxa de perda de pacotes nas comunicações. Adequar uma aplicação em tempo real às mais variadas situações da Internet se torna um desafio.

- **Máquinas:** Diferentes configurações de computadores e diversos dispositivos, como telefones celulares, estão conectados entre si através da Internet. Quando se tem prévio conhecimento das máquinas, pode-se otimizar uma aplicação para determinados usuários. Porém, em aplicações como jogos e museus virtuais, a informação sobre as configurações dos clientes pode não estar acessível, dificultando a forma de distribuir as tarefas entre cliente e servidor.

2.2 Cenários

A visualização de conteúdo 3D na Internet pode ser realizada de diferentes maneiras, dependendo da finalidade da aplicação em questão e da capacidade computacional do

computador do usuário que visualiza os dados disponibilizados na Internet. Basicamente três abordagens podem ser utilizadas para visualização de conteúdo 3D no cliente:

1. Visualização de imagens estáticas pré-renderizadas de modelos 3D.
2. Visualização de imagens estáticas renderizadas em um servidor em tempo real a partir de modelos 3D, de acordo com requisições.
3. Visualização de modelos 3D.

A visualização que utiliza imagens destaca-se pela capacidade de exibir conteúdo com mais detalhes [28], no entanto a interatividade é limitada. O modelo 3D encontra-se em um servidor, sendo esta abordagem recomendada apenas para efeitos de visualização de detalhes dos modelos 3D, sem possibilitar a interação com os mesmos.

Já a visualização que utiliza modelos 3D possibilita interatividade com o conteúdo visualizado, pois o cliente manipula o modelo 3D diretamente. Em compensação, o realismo pode ser prejudicado, pois renderizar um modelo 3D é computacionalmente mais pesado, o que infere que o modelo 3D visualizado pelo usuário deve ter o nível de detalhes reduzido.

Com base nestas abordagens para visualização de conteúdo 3D e considerando que a renderização de modelos 3D pode envolver cálculos complexos, a forma de distribuir as tarefas entre o servidor e os clientes torna-se um problema que deve ser tratado com grande importância [39]. A disponibilização de conteúdo 3D pode ser dividida em quatro categorias [18, 25, 39], como mostra a Figura 2.1. Cada uma das categorias constitui um cenário, que diferem nas ações executadas pelo cliente e pelo servidor.

No cenário 1 temos apenas visualização de imagens estáticas pré-renderizadas e/ou imagens renderizadas em tempo real. Neste cenário podemos utilizar um servidor para armazenar os modelos 3D [19, 38], e em seguida o cliente pode acessar o servidor para obter os dados e visualizá-los. Nos demais cenários temos visualização de imagens e/ou

visualização de modelos 3D. Detalhes sobre os cenários e exemplos de alguns projetos que os utilizam podem ser vistos nas próximas seções.

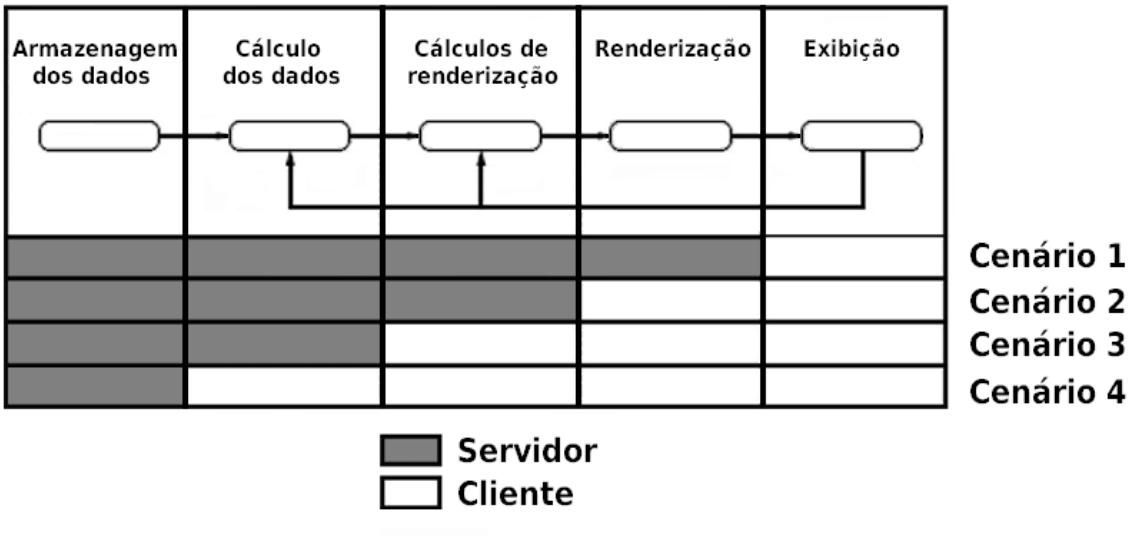


Figura 2.1: Cenários de distribuição de tarefas entre cliente e servidor.

2.2.1 Cenário 1

Consiste em dedicar um servidor robusto, com recursos computacionais poderosos, para armazenar os modelos 3D e fazer todos os cálculos complexos [39]. Esse método é amplamente utilizado para visualização e fornece uma imagem para o cliente, a qual corresponde ao resultado da renderização remota. Nesse caso, o cliente não precisa ter recursos computacionais poderosos para visualizar o conteúdo desejado.

A técnica baseada em imagens (Image-based Rendering) descreve uma cena no cliente utilizando apenas imagens ao invés de dados 3D (geometria do objeto, propriedade dos materiais, iluminação, entre outros). Em [38] é utilizado um sistema de renderização remota baseada em imagens, em que modelos 3D são renderizados no servidor e imagens são geradas e comprimidas, sendo enviadas ao cliente que as descompacta e exibe na tela.

Engel *et al.* [11] apresentam um sistema para visualização interativa de conteúdo 3D com aceleração 3D remota para redes de banda baixa, utilizando conexão de 64 kBit ISDN (Rede Digital de Serviços Integrados). Uma máquina poderosa é utilizada como servidor

e ocorre apenas a transferência de imagens para o cliente. O servidor é descrito como uma aplicação Open Inventor ou Cosmo3D, que transmite imagens comprimidas para a aplicação cliente desenvolvida na linguagem Java.

É importante observar que este cenário, além de considerar que as redes de transmissão podem ter baixas taxas de transferência, também provê a segurança dos modelos 3D. Isso acontece pois apenas o servidor pode acessar os modelos 3D de alta resolução[1]. Sistemas semelhantes, em que os recursos do cliente são limitados e a potência da rede é baixa, podem ser observados em [13, 20, 35].

Recentemente uma nova plataforma de jogos sob demanda foi anunciada na Game Developers Conference de 2009, chamada de OnLive[2]. O sistema armazena os jogos em um servidor remoto e aguarda requisições dos usuários. Ao receber uma requisição, o jogo é sincronizado, renderizado e o resultado da renderização é entregue para o usuário através da Internet. Mesmo com os grandes avanços tecnológicos para computadores pessoais, grandes empresas produtoras de jogos, como Eletronic Arts, Take-Two, Codemasters, Warner Bros entre outras, estão disponibilizando seus jogos através do serviço do OnLive, que é um serviço pago.

O serviço é visto como um grande competidor no mercado de vídeo-games, já que um computador simples, que possua os requisitos mínimos para reprodução de vídeos, pode ser utilizado para acesso ao sistema, sendo possível jogar qualquer título disponível no servidor. No entanto, ótimas conexões de banda larga são requeridas. Conexões de 1.5 Mbps permitem imagens de qualidade analógica, enquanto conexões de 4 a 5 Mbps podem fornecer imagens de alta definição. Outras aplicações recentes, como Gaikai[3] e Otoy[4] seguem a mesma idéia de disponibilizar jogos sob demanda com servidores de renderização remota.

[1]Modelos 3D de alta resolução possuem uma grande quantidade de polígonos.
[2]http://www.onlive.com
[3]http://www.gaikai.com
[4]http://www.otoy.com

2.2.2 Cenário 2

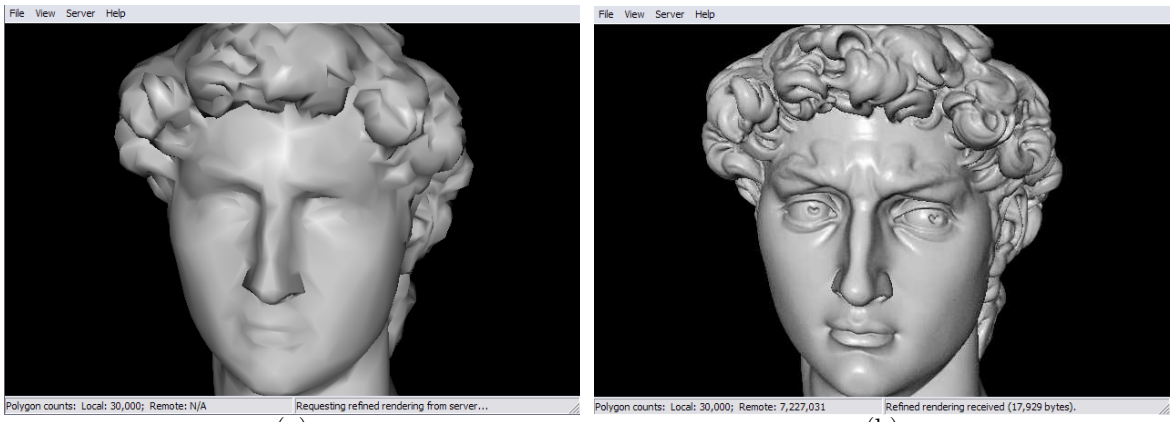
A fim de utilizar as capacidades tanto do servidor quanto do cliente, as tarefas de renderização são distribuídas uniformemente [39]. Pesquisadores utilizam essa abordagem para equilibrar a carga de trabalho entre o cliente e o servidor, diminuindo o uso da rede de comunicação e melhorando o desempenho no cliente [7, 38].

No "The Digital Michelangelo Project" [23] foram digitalizadas grandes esculturas feitas por Michelangelo. Neste projeto, os autores comentam sobre o problema da distribuição não autorizada dos modelos 3D, já que podem possuir direitos autorais, e da reconstrução física dos objetos se houver o acesso direto aos dados dos modelos 3D de alta resolução. A solução utilizada no projeto foi desenvolver um sistema de renderização remota [19], com o intuito de evitar que modelos 3D de alta resolução sejam disponibilizados para os usuários.

O sistema desenvolvido utiliza uma arquitetura cliente-servidor, no qual os modelos 3D de alta resolução ficam armazenados no servidor e o cliente utiliza apenas um modelo 3D simplificado para interação. A ferramenta de visualização "ScanView"(cliente do sistema) permite ao usuário interagir com o modelo 3D, o qual pode ser rotacionado, afastado, aproximado, entre outras operações.

Ao parar de interagir com o modelo 3D, o usuário recebe uma imagem de alta resolução do servidor, podendo observar melhor os detalhes do objeto. Essa imagem gerada no servidor é a renderização do modelo 3D de alta resolução no mesmo ponto de vista em que está sendo visualizado o modelo 3D simplificado no cliente (Figura 2.2).

Luke e Hansen [25] apresentam um sistema flexível de renderização remota que distribui as tarefas entre o servidor e o cliente baseado em banda, latência e poder computacional do cliente e do servidor. O primeiro modelo de renderização do sistema baseado em fluxo de imagens realiza todos os cálculos da renderização apenas no servidor, enquanto que o cliente apenas exibe as imagens na tela. Esse modelo considera que o usuário não possui recursos computacionais suficientes para realizar a visualização 3D localmente.



(a) (b)

Figura 2.2: Exemplo de visualização usando o sistema proposto no "The Digital Michelangelo Project": (a) modelo 3D de baixa resolução renderizado localmente no cliente e (b) imagem do modelo 3D de alta resolução renderizado no servidor e transmitida ao cliente via Internet.

O segundo modelo de renderização do projeto considera o cenário contrário, enviando o dado 3D para ser renderizado no cliente e encerrando a conexão logo em seguida. Por fim, um terceiro modelo divide o trabalho entre cliente e servidor. O servidor envia uma malha 3D simplificada correspondente apenas à vista que está visível e uma imagem para ser utilizada como textura na renderização local do cliente.

O Virtual Inspector [8] é uma ferramenta que permite aos usuários inspecionar grandes modelos 3D em tempo real. A ferramenta, voltada aos usuários das áreas da Preservação Digital, provê ao usuário uma abordagem de interação fácil e intuitiva. Os objetivos do trabalho foram: obter performance com computadores de baixo custo; interação simples e intuitiva para usuários iniciantes; alta qualidade na renderização dos grandes modelos 3D; permitir aos profissionais da Preservação Digital associar informações ao modelo 3D; permitir acesso local e remoto; e proteção aos dados 3D. A visualização na Internet pode ser feita utilizando um navegador *web* próprio desenvolvido para esse fim.

O suporte à técnica de renderização remota também está presente no Virtual Inspector, assim como proposto em [19]. O computador local que utiliza a ferramenta recebe o modelo 3D com resolução reduzida para apoiar a interação do usuário. Quando um ponto de vista é selecionado, o cliente faz uma solicitação a um servidor o qual envia uma imagem do modelo 3D de alta resolução renderizado com os parâmetros de exibição no

cliente. Os autores comentam que um inconveniente desse tipo de abordagem é o tempo de latência e carga da rede, além da necessidade de configurar uma grande infra-estrutura de renderização paralela (*render farm*) adequada para suportar a carga prevista.

Os experimentos com a técnica de renderização remota utilizaram um servidor com sistema operacional Ubuntu Linux, um processador Athlon 64 3500+, 2GB RAM, NVIDIA GeForce 6600, 256MB de memória, e conexão da Internet de 1.2 Mbps. Para realização dos testes foram utilizados o modelo 3D da Catedral de Pisa (aproximadamente 390.000 faces) e dois modelos 3D de Davi (aproximadamente 50.000 faces cada um). Os resultados foram os seguintes:

Catedral de Pisa: Tempo de requisição + tempo da renderização remota + tempo para realizar a compressão da imagem + tempo de transmissão + tempo de descompressão no cliente + tempo de exibição da imagem na tela: Entre 750 e 2000 milisegundos. A renderização remota no lado do servidor demora entre 100 a 500 milisegundos dependendo da vista.

Estátua de Davi: Com o mesmo processo descrito acima, o tempo total fica entre 150 e 650 milisegundos.

2.2.3 Cenários 3 e 4

No cenário 3, o servidor pode armazenar e transmitir os dados 3D progressivamente ao cliente enquanto no cenário 4 o servidor apenas armazena os dados 3D e transmite. Esses métodos realizam a maior parte da renderização dos modelos 3D no cliente [10, 32]. Essa abordagem pode tirar o máximo poder da capacidade do *hardware* gráfico dos clientes. Com a evolução do *hardware* dos usuários, especialmente em dispositivos móveis, esse modelo pode ser adotado por mais sistemas e usuários [39].

2.3 Características de um Sistema de Renderização Remota

Alguns estudos já foram elaborados em relação a qual cenário seria adequado para disponibilizar conteúdo 3D na Internet, mantendo os possíveis direitos autorais das obras digitalizadas. Um dos estudos mais aprofundados foi feito por Koller *et al.* [19], mostrando quais ataques podem ser efetuados nos modelos 3D e concluindo que a melhor solução para proteger esses dados consiste em utilizar um sistema de renderização remota.

Como observado em [38], para utilizar eficientemente a visualização para pesquisas e estudos científicos na Internet, um sistema de visualização deve possuir as seguintes características:

- O modelo 3D de alta resolução deve estar no servidor e não pode ser transferido para o cliente.
- O mínimo de banda da rede deve ser consumido em cada comunicação.
- Modestos recursos computacionais são esperados no cliente.

Essas características estão de acordo com a solução de utilizar a renderização remota para visualização de modelos 3D na Internet. Outra característica, descrita em [7], é de que um sistema para visualização de dados 3D deve permitir que, mesmo usuários leigos com computadores não potentes, possam visualizar um grande número de modelos 3D complexos. Em [7] foi seguida uma abordagem de renderização remota semelhante ao cliente-servidor apresentado em "The Digital Michelangelo Project" [19].

A abordagem de renderização remota apresenta algumas vantagens: baixa exigência computacional para renderização no cliente; não há limite de resolução do modelo 3D no servidor; proteção dos modelos 3D de alta resolução. Outra vantagem é que a renderização remota permite que o servidor tenha modelos 3D em várias resoluções, as quais podem ser alteradas quando necessário sem necessidade de alterações no cliente. Os métodos de renderização utilizados no servidor também podem mudar, otimizando a renderização

com algoritmos mais sofisticados, sem necessidade de mudanças no cliente [7]. Podemos também ter um *cache* preditivo (Seção 2.3.3) para diminuir o tempo de resposta de visualização do conteúdo para o usuário.

A renderização remota também agrega segurança ao sistema, tanto aos modelos 3D armazenados no servidor quanto ao conteúdo 3D enviado ao cliente. Medidas de segurança podem ser incorporadas, como por exemplo implementar técnicas de defesa para proteger o servidor de possíveis ataques. Informações sobre como garantir a segurança nesses sistemas são apresentadas na próxima seção.

2.3.1 Segurança

Com a renderização remota, os modelos 3D de alta resolução ficam armazenados no servidor do sistema com segurança, mas ainda assim os modelos 3D podem sofrer ataques no cliente. Os possíveis ataques aos modelos 3D que foram estudados em [19] são:

- **Engenharia reversa no arquivo do modelo 3D:** Os arquivos de modelos 3D enviados para o usuário podem ser criptografados, porém a chave de criptografia pode ser descoberta. Tais codificações podem ser quebradas a partir do uso de engenharia reversa, deixando o arquivo do modelo 3D vulnerável.

- **Adulterar a aplicação de visualização:** Técnicas podem ser usadas para obter informações privilegiadas dos visualizadores 3D, como por exemplo os dados do modelo 3D.

- **Adulterar o *driver* gráfico:** Os dados 3D se tornam vulneráveis quando são utilizados pelo *driver* da placa de vídeo. O ataque pode consistir em substituir o *driver* por um outro malicioso e capturar os dados do modelo 3D.

- **Reconstrução a partir do *buffer*:** Os dados do *buffer* que contém as informações mostradas na tela podem ser capturados e utilizados para reconstrução da geometria do modelo 3D.

- **Reconstrução da imagem final de exibição:** A imagem exibida para o usuário não está livre de ataques, mesmo que todo o processo anterior à exibição tenha sido protegido. As imagens podem ser utilizadas como entrada para um sistema de reconstrução 3D.

Apesar desses possíveis ataques, a utilização da renderização remota pode ser considerada uma abordagem segura para os dados 3D. Para isso, é necessário incorporar medidas de segurança ao sistema. No caso de modelos 3D de baixa resolução enviados para interação no cliente, para os quais a disponibilização não preocupa os proprietários [19], pode-se inserir marcas d'água. Quanto ao servidor, algumas técnicas de defesas podem ser usadas, como as que são citadas na seção a seguir.

2.3.2 Defesas do Servidor

A renderização remota pode ser vulnerável à ataques de reconstrução a partir das imagens finais de exibição. Os modelos 3D se tornam sensíveis à reconstrução, pois pode-se especificar exatamente os parâmetros utilizados para gerar as imagens e roubar os dados. Sendo assim, é necessário defender o sistema de renderização remota dos ataques. Abaixo uma breve descrição de algumas técnicas vistas em [19]:

- **Defesa baseada por sessão:** Cada cliente é único. Ações suspeitas, como várias requisições numa sequência de vistas, podem ser detectadas e atribuídas a um usuário específico.

- **Criptografia:** Modelos 3D de baixa resolução e a requisição do cliente para o servidor podem ser criptografados. Utilizar a criptografia como único meio de defesa pode não ser válido, mas incluso em um sistema de renderização remota pode ser eficiente.

- **Limitações sobre os pedidos de renderizações:** Muitas requisições de apenas uma determinada região podem ser consideradas um ataque. Limitar seguidos acessos à regiões importantes dos modelos 3D pode evitar a reconstrução dos mesmos.

- **Perturbações e distorções:** Cada requisição deve gerar uma imagem diferente, mesmo que os parâmetros vindos do cliente sejam os mesmos. Altera-se, por exemplo, a iluminação requisitada e adicionam-se alguns ruídos.

- ***Firewall*:** A utilização de um *firewall* pode regular o tráfego de dados entre redes distintas, impedindo a transmissão e/ou recepção de acessos nocivos ou não autorizados ao servidor.

2.3.3 Cache Preditivo

Para diminuir o tempo entre a requisição do cliente e a resposta do servidor, algumas técnicas podem ser utilizadas. Uma comumente utilizada em sistemas de renderização remota baseados em imagens é chamada de 3D Warping [27]. Nessa técnica, as vistas são previstas através de uma transformação geométrica, capaz de mapear uma imagem de entrada para uma imagem de saída representando uma vista arbitrária. A imagem de entrada contém informações de profundidade de cada *pixel*, além de parâmetros da câmera.

Com a utilização dessa técnica alguns problemas podem ocorrer, como por exemplo a presença de buracos e artefatos. Para a visualização realista voltada à Preservação Digital, é necessário garantir a fidelidade visual dos modelos 3D. Por isso, outra abordagem que pode ser utilizada para evitar estes problemas é prever as vistas de acordo com as interações do usuário com o modelo 3D. Dada uma operação, o sistema prevê as futuras vistas, as quais são representadas por imagens e armazenadas em um *cache* preditivo.

Em [9], as interações dos usuários são caracterizadas a fim de enviar os modelos 3D progressivamente para a visualização. Foi realizado um experimento com 37 usuários, os quais interagiram com 9 modelos 3D. Verificou-se que em determinados cenários, as

ações dos usuários são altamente previsíveis, indicando que técnicas de predição podem ser muito úteis.

Em um determinado cenário, os autores selecionaram três modelos 3D de diferentes complexidades disponibilizados no repositório de digitalização 3D de Stanford[5]: Happy Buddha, Dragon e Thai Statue. O experimento realizado considerou um cenário do mundo real, onde clientes podem realizar compras em uma loja de antiguidades. A loja *on-line* possui três produtos que correspondem aos três modelos 3D selecionados para o teste, os quais podem ser visualizados pelos clientes. Esse experimento contou com a participação de 25 pessoas, as quais foram previamente orientadas sobre os comandos do teclado para interagir com os modelos 3D. As operações disponíveis foram: rotação no sentido horário e anti-horário; translação (cima, baixo, direita, esquerda); aproximar e afastar.

Os testes mostraram que algumas operações ocorrem com bastante frequência para todos os modelos 3D, como por exemplo rotação no eixo Y e aproximar. Para este último, foi observado que ocorre com mais frequência que o afastar, indicando também que os usuários tendem a ampliar para uma distância confortável para depois realizar outras operações.

Notou-se também que a distribuição das operações está relacionada com as regiões do modelo 3D. Por exemplo, a probabilidade da operação aproximar ser utilizada é maior na região frontal do modelo 3D. Já em objetos com orientação horizontal, os usuários realizam com maior frequência a operação de rotação no eixo X. No geral, os resultados indicaram que o ponto de vista do modelo 3D afeta a probabilidade de predição da próxima operação, sendo dependente do tamanho e forma do objeto.

[5]http://graphics.stanford.edu/data/3Dscanrep/

CAPÍTULO 3

SISTEMA DE RENDERIZAÇÃO REMOTA

O sistema de renderização remota desenvolvido aproveita as vantagens das abordagens de visualização mostradas na Seção 2.2, enquadrando-se nos cenários 1 e 2 entre os possíveis cenários de formas de distribuição de conteúdo 3D. O sistema segue a abordagem cliente-servidor semelhante aos projetos vistos em [19, 38], composto por um servidor de renderização remota, que contém os modelos 3D de alta resolução, e pelo cliente que acessa o servidor para obter os modelos 3D de baixa resolução e/ou imagens.

A visualização no cliente ocorre de acordo com a configuração do computador do usuário. Foram desenvolvidos três visualizadores que utilizam a renderização remota, permitindo que qualquer computador com acesso à Internet, independente da sua configuração, consiga acessar o conteúdo 3D disponibilizado. Computadores com placa de vídeo sem aceleração 3D poderão visualizar imagens de modelos 3D, enquanto computadores com placa de vídeo com aceleração 3D poderão visualizar tanto modelos 3D quanto imagens, observando assim mais detalhes dos objetos.

Como mostra a Figura 3.1, o cliente 1 interage apenas com modelos 3D localmente, possuindo placa de vídeo com aceleração 3D. O cliente 2 também possui placa de vídeo com aceleração 3D e visualiza tanto modelos 3D localmente, quanto imagens obtidas com a renderização remota. O cliente 3 possui placa de vídeo sem aceleração 3D, visualizando apenas imagens recebidas do servidor de renderização remota.

O uso do sistema de renderização remota permite a existência dos clientes 2 e 3. Detalhes sobre os visualizadores desenvolvidos são explicados na Seção 3.1. As inicializações necessárias para garantir o correto funcionamento do sistema, assim como o funcionamento do servidor de renderização remota são apresentados neste capítulo.

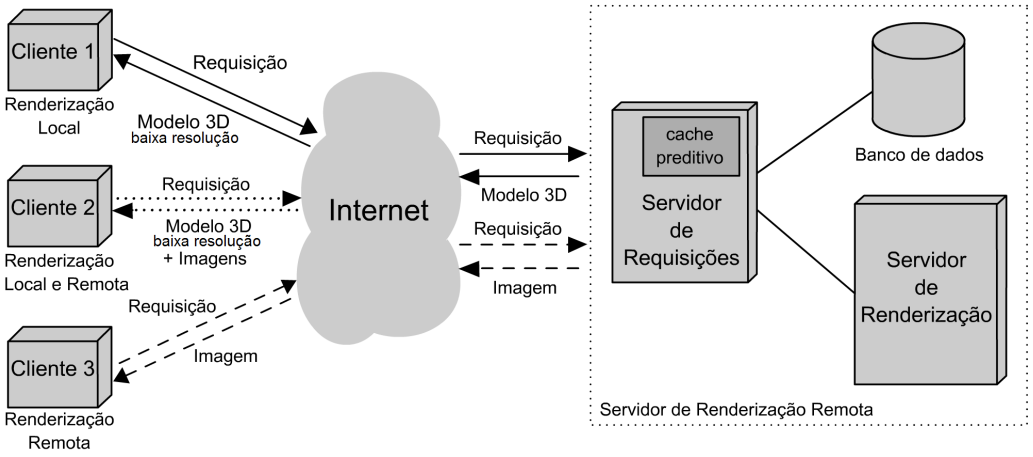


Figura 3.1: Arquitetura completa do sistema de renderização remota. O Cliente 1 apenas manipula modelos 3D de baixa resolução. Os clientes 2 e 3 utilizam o sistema de renderização remota para visualizar modelos 3D e/ou imagens.

3.1 Visualizadores

Para permitir que o conteúdo disponibilizado seja acessível por qualquer usuário, não devemos exigir requisitos mínimos para o uso do sistema além do acesso à Internet. Com esse pensamento, foram desenvolvidos três visualizadores para o sistema de renderização remota, não restringindo o acesso apenas a usuários que possuem computadores com placa de vídeo com aceleração 3D. Diferentemente de projetos como Michelangelo Digital [23] e Virtual Inspector [8], todos os visualizadores desenvolvidos são acessados através de um navegador *web*, aumentando a acessibilidade ao conteúdo disponibilizado.

3.1.1 Visualizador 1: Modelo 3D e Imagens

Este visualizador enquadra-se no cenário 2 de distribuição de tarefas entre cliente e servidor (Seção 2.2.2) atendendo aos usuários que possuem computador com placa de vídeo com aceleração 3D e assim, podem visualizar e interagir com modelos 3D. É semelhante ao visualizador visto em [19], garantindo interatividade e acesso aos detalhes dos modelos 3D.

O usuário recebe o modelo 3D de baixa resolução para interação, como mostra a Figura 3.2, e ao parar de movimentá-lo, o cliente reúne os parâmetros da visualização necessários e envia a requisição ao servidor de renderização remota. O servidor renderiza

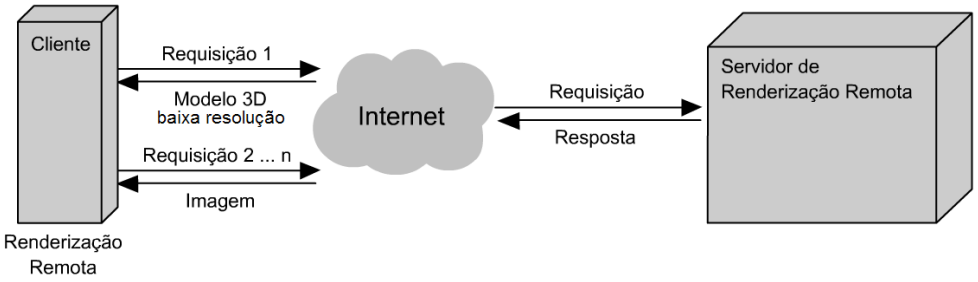


Figura 3.2: Arquitetura do sistema em relação ao visualizador 1. O cliente recebe um modelo 3D de baixa resolução do servidor para interação e nas próximas requisições recebe imagens com detalhes do modelo 3D escolhido.

o modelo 3D de alta resolução com os parâmetros de visualização recebidos e envia uma imagem para o cliente. Ao ser recebida, a imagem do modelo 3D de alta resolução é sobreposta ao modelo 3D de baixa resolução. Um exemplo pode ser visto na Figura 3.3. Mais detalhes sobre o servidor de renderização são vistos na Seção 3.4.

(a) (b)

Figura 3.3: Exemplo de visualização no cliente de um modelo 3D de uma estátua presente no gabinete do reitor da Universidade Federal do Paraná: (a) modelo 3D de baixa resolução e (b) imagem do modelo 3D de alta resolução.

O projeto do Museu Virtual 3D do grupo de pesquisa IMAGO (Capítulo 4) possui dois visualizadores 3D para navegadores *web*: o Plugin IMAGO [31] e o IMAGO 3D Viewer. Primeiramente foi desenvolvido o Plugin IMAGO, na linguagem C/C++, como alternativa para visualizadores VRML, os quais não se mostravam adequados para visualização de acervos na Preservação Digital [29]. Posteriormente, foi disponibilizado o IMAGO 3D Viewer, desenvolvido na linguagem Java e JOGL (biblioteca que permite utilizar o OpenGL com a linguagem Java), incorporando novos recursos de *interface* e permitindo uma visualização mais simples e interativa para os usuários iniciantes. Mais detalhes sobre o funcionamento desses dois visualizadores podem ser vistos em [29].

Para permitir a visualização com a técnica de renderização remota, um módulo para esse fim foi incorporado aos dois visualizadores 3D. Os visualizadores, além de permitir a interação com os modelos 3D em navegadores *web*, também comportam-se como clientes do sistema. Eles são responsáveis por enviar os parâmetros da visualização para o servidor, receber a resposta e exibir a imagem sobre o modelo 3D. A visualização com renderização remota ocorre em paralelo à visualização com renderização local, seguindo os seguintes passos:

1. **Inspeção das operações feitas pelo usuário:** Um módulo do visualizador 3D verifica quais operações o usuário está realizando para manipulação dos modelos 3D. Quando o modelo 3D está sendo exibido na tela, após alguns milisegundos de inatividade de operações, os próximos passos da renderização remota são iniciados. Caso uma imagem esteja sendo exibida na tela e alguma operação seja iniciada pelo usuário, o modelo 3D é renderizado na tela.

2. **Definição da requisição:** Todos os parâmetros de visualização necessários para o servidor são capturados para a definição da requisição, exemplificado na Figura 3.4. O tipo da requisição indica qual é o visualizador utilizado.

Tipo da Requisição	Nome do Modelo 3D	Largura e Altura	Coordenadas (x, y, z)	Wireframe	Textura	Iluminação

Figura 3.4: Parâmetros utilizados no visualizador 1 enviados para o servidor de renderização remota.

3. **Comunicação com o servidor:** O módulo de comunicação envia a requisição ao servidor e aguarda uma imagem como resposta. O protocolo HTTPS é utilizado para realizar a comunicação entre o visualizador (Java Applet) e o servidor de requisição (Java Servlet).

4. **Sobreposição da imagem:** A imagem recebida pelo cliente é desenhada na tela de exibição da aplicação, sobrepondo o modelo 3D.

5. Volta ao passo 1.

Os testes realizados, que podem ser vistos na Seção 4.2, mostraram o bom funcionamento da técnica de renderização remota nos dois visualizadores 3D do IMAGO. As vantagens oferecidas pela linguagem Java, como por exemplo os recursos para realizar a comunicação na Internet e a manipulação de imagens, além do aumento da produtividade devido à característica de portabilidade da mesma, indicaram o IMAGO 3D Viewer como melhor visualizador de modelos 3D para ser utilizado nesse sistema.

3.1.2 Visualizador 2: Imagens

O visualizador 2 é utilizado por usuários que possuem computador com placa de vídeo sem aceleração 3D. Se enquadra no cenário 1 (Seção 2.2.1), em que o usuário pode visualizar apenas dados 2D, ou seja, imagens renderizadas de modelos 3D que o servidor envia, como mostra a arquitetura do visualizador 2 na Figura 3.5. O visualizador 2 e o visualizador 3 (Seção 3.1.3) garantem que qualquer pessoa, que possua um computador com acesso à Internet, consiga visualizar o conteúdo disponibilizado por um museu virtual por exemplo, mesmo que através de imagens. Este visualizador foi desenvolvido com a tecnologia Java Applets, permitindo ao usuário visualizar imagens de modelos 3D através de uma *interface* simples e intuitiva.

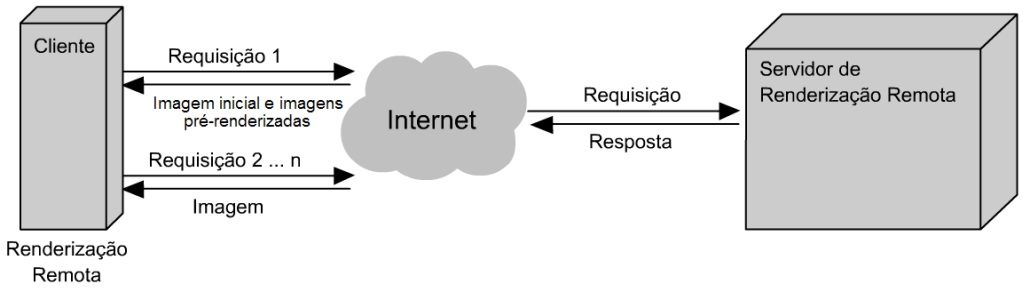


Figura 3.5: Arquitetura do sistema proposto em relação ao visualizador 2. O cliente recebe uma imagem inicial e imagens pré-renderizadas para visualização e nas próximas requisições recebe imagens com detalhes do modelo 3D escolhido.

Após o usuário escolher o modelo 3D que deseja visualizar, o visualizador recebe uma imagem de referência (640 x 480 *pixels*) que é exibida na tela como imagem inicial. A imagem de referência é uma imagem renderizada do modelo 3D de alta resolução no

servidor. Para auxiliar a visualização, a *interface* dispõe de um painel de navegação no qual é possível visualizar imagens menores pré-renderizadas (220 x 165 *pixels*) do modelo 3D escolhido. Essas imagens são anexadas em um arquivo no formato zip e enviadas na chamada do visualizador na página HTML. Mais detalhes sobre a geração das imagens pré-renderizadas utilizadas são vistos na Seção 3.2.2.

A visualização das imagens é de forma interativa, utilizando uma *interface* própria para este visualizador. Com o painel de navegação, o usuário pode pré-visualizar, em tempo real, a posição desejada do modelo 3D antes de realizar a requisição da imagem para o servidor. Após o usuário escolher as opções desejadas, uma requisição é enviada ao servidor que retorna uma imagem, esta sim, utilizando todas as opções escolhidas pelo usuário (informações de textura, iluminação, entre outras). A imagem recebida é exibida no lado direito da tela (Figura 3.6).

Figura 3.6: *Interface* para o visualizador 2. Imagem de referência inicial pode ser vista a direita. Imagens menores de interação estão dispostas no canto superior esquerdo. O modelo 3D é o fóssil de um Protocyon (animal que vivia durante o Pleistoceno) e que pertence ao Museu de Ciências Naturais da UFPR.

A requisição (Figura 3.7) enviada para o servidor de renderização remota possui o formato similar à descrita na Seção 3.1.1, porém utiliza alguns parâmetros diferentes. O tipo da requisição, o nome do modelo 3D, o ângulo theta (correspondente à rotação no eixo X), ângulo fi (correspondente à rotação no eixo Y) e opções de *wireframe*, textura e *zoom* são utilizados.

Tipo da Requisição	Nome do Modelo 3D	Ângulo theta	Ângulo fi	Wireframe	Textura	Iluminação	Zoom

Figura 3.7: Parâmetros utilizados no visualizador 2 enviados para o servidor de renderização remota.

3.1.3 Visualizador 3: Imagens

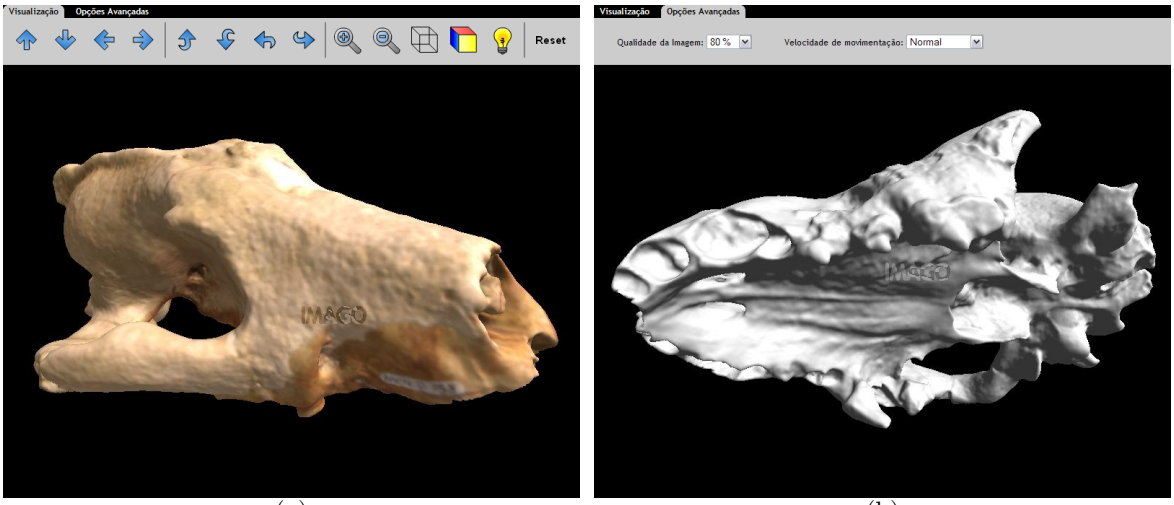
Após a construção do visualizador 2 observou-se que era possível facilitar o acesso do conteúdo aos usuários. Sendo assim, foi disponibilizada uma outra abordagem para visualização que utiliza a tecnologia Java Server Pages (JSP) e Asynchronous JavaScript and XML (Ajax). Essa abordagem permite a visualização das imagens sem a necessidade de realizar qualquer instalação, como por exemplo do Java Runtime Environment (JRE) ou qualquer outro tipo de *plugin*. O Ajax utiliza um objeto chamado XMLHttpRequest[1] para efetuar pedidos através do protocolo HTTP, que ao receber a resposta do servidor, realiza as alterações necessárias na página *web*.

Semelhante ao visualizador 2 no propósito de permitir que qualquer computador possa visualizar o conteúdo disponibilizado, o visualizador 3 envia apenas imagens para visualização no cliente, também pertencendo ao cenário 1 das formas de distribuição de tarefas (Seção 2.2.1). Após o usuário escolher um modelo 3D que deseja visualizar, uma página *web* é exibida contendo a *interface* de visualização. A *interface* (Figura 3.8) apresenta uma imagem no centro da tela e dispõe de botões para realizar a movimentação do modelo 3D. Ao selecionar um dos botões, uma requisição é enviada ao servidor que responde com uma imagem do modelo 3D de alta resolução renderizado.

A imagem recebida é exibida no centro da tela sem o recarregamento completo da página *web*, graças à utilização de JSP em conjunto com tecnologia Ajax. A imagem gerada pelo servidor é repassada para um servidor de requisições (apresentado na Seção 3.3) e é armazenada na memória. Para a exibição no lado do cliente é necessário converter a imagem para o sistema numérico Base64[2], pois o parâmetro *src* na tag *img* de exibição do

[1] O XMLHttpRequest permite realizar uma comunicação assíncrona com o servidor, efetuando atualizações sem recarregar a página por completo.

[2] Base64 é um sistema numérico que utiliza 64 caracteres. É representado utilizando-se apenas caracteres ASCII, sendo utilizado, entre outras finalidades, para codificação de transferência de e-mails.



(a) (b)

Figura 3.8: *Interface* para o visualizador 3. Apenas a imagem do modelo 3D é atualizada a cada requisição do usuário: (a) Aba de movimentação e opções como textura e iluminação do modelo 3D, (b) Aba de opções da qualidade da imagem a ser recebida e velocidade de movimentação.

HTML indica um *link* para imagem. A transformação utilizada da imagem para Base64 demora, em média, 15 milisegundos.

Além das opções básicas, o usuário pode escolher a qualidade da imagem a ser recebida e a velocidade de movimentação, que influencia no valor dos ângulos utilizados para renderizar os modelos 3D. A requisição enviada para o servidor é semelhante à do visualizador 1, como mostra a Figura 3.9. As duas diferenças são: a adição de um parâmetro que indica a qualidade da imagem a ser recebida e exibida para o usuário; retirada das informações sobre a movimentação da luz incidente sobre o modelo 3D.

Tipo da Requisição	Nome do Modelo 3D	Largura e Altura	Coordenadas (x, y, z)	Wireframe	Textura	Iluminação	Qualidade da imagem

Figura 3.9: Parâmetros utilizados no visualizador 3 que são enviados para o servidor.

3.2 Inicialização do Sistema

Antes de iniciar o funcionamento do sistema de renderização remota, é necessário realizar algumas etapas de preparação do sistema. Os modelos 3D que estarão disponíveis no sistema são selecionados, considerando aspectos de segurança e realismo. A Seção 3.2.1

apresenta a etapa de simplificação desses modelos 3D, os quais serão disponibilizados aos usuários no visualizador 1. A geração das imagens pré-renderizadas utilizadas no visualizador 2 é apresentada na Seção 3.2.2. Outros fatores relacionados à inicialização do sistema são apresentados na sequência.

3.2.1 Simplificação dos Modelos 3D

Para a visualização em um sistema de renderização remota é necessário gerar modelos 3D de baixa resolução para serem enviados e renderizados no cliente. A primeira abordagem adotada foi gerar os modelos 3D durante o processo de reconstrução 3D [36], utilizando um *script* de entrada na etapa de integração de vistas, no qual é escolhido o tamanho do *voxel* desejado.

Essa abordagem possui duas grandes desvantagens. A integração das vistas ocorre praticamente na metade do processo de reconstrução 3D, obrigando a execução de quase todas as etapas do processo para a criação de um único modelo 3D de baixa resolução. O modelo 3D gerado possui uma malha com triângulos de tamanhos iguais em distribuição, desvalorizando regiões que possuem detalhes e que, consequentemente, necessitam de triângulos menores.

Posteriormente, visto que essa abordagem não era produtiva para o sistema, notou-se a necessidade de utilizar métodos de simplificação para gerar malhas 3D com diferentes níveis de detalhes. Uma malha 3D simplificada é uma malha representada por uma quantidade menor de triângulos, mas que consegue manter as partes mais salientes da geometria, as quais capturam a forma do objeto. A simplificação dos modelos 3D busca reduzir o custo de processamento sem perda significativa na qualidade visual do objeto [24].

Neste trabalho utilizamos o algoritmo proposto por Garland e Heckbert [16], o qual é capaz de produzir modelos 3D simplificados de forma rápida, preservando a forma geral da superfície do objeto. O algoritmo utiliza o princípio de decimação por contração de arestas, ou seja, uma aresta é eliminada e substituída por um novo vértice que introduz o

menor erro em relação à malha. A Figura 3.10 demonstra esse processo, onde a aresta $v_i v_j$ é substituída pelo vértice v. Mais detalhes sobre o algoritmo podem ser vistos em [15].

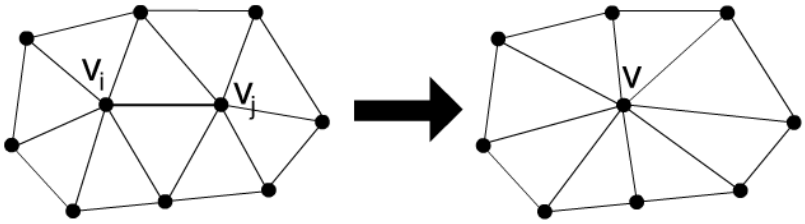


Figura 3.10: Contração de aresta utilizando o algoritmo proposto por Garland e Heckbert.

Os modelos 3D simplificados apresentam boa qualidade visual, preservando as características mais importantes do objeto (Figura 3.11). Com o algoritmo utilizado, é possível escolher a quantidade de faces desejada para o modelo 3D simplificado, permitindo qualificar os melhores, e ao mesmo tempo, menores modelos 3D para serem enviados ao cliente.

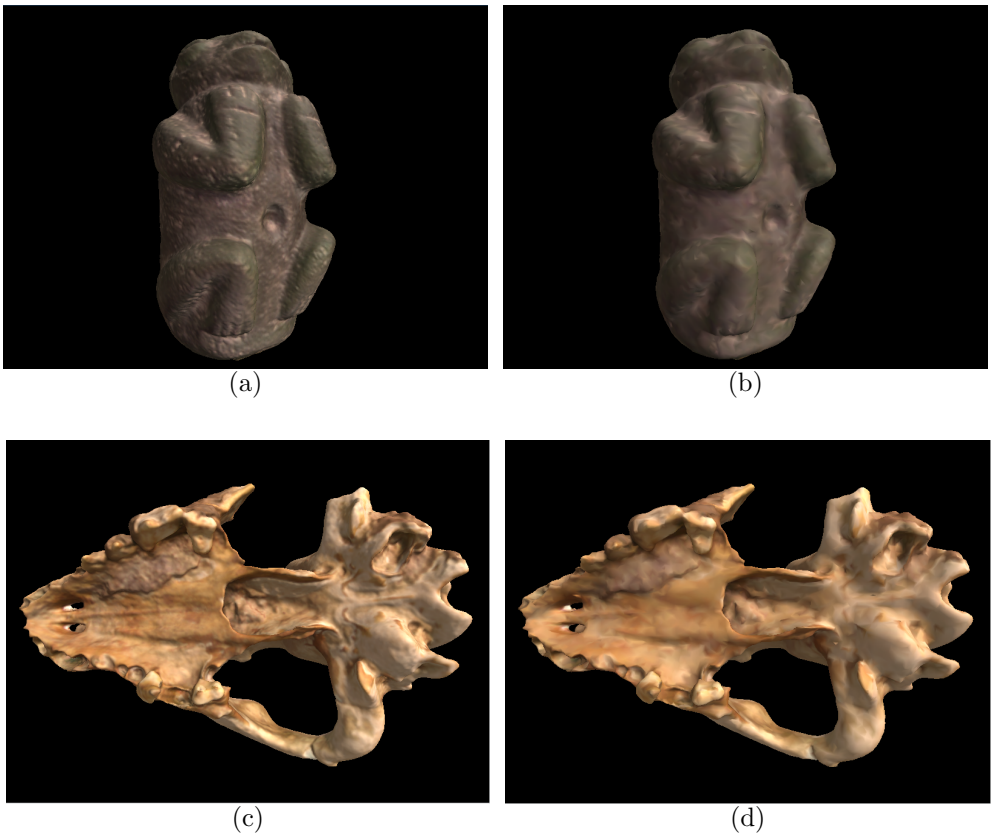


Figura 3.11: Simplificação dos modelos 3D: (a) Alamito com 342.520 faces , (b) Alamito com 17.126 faces, (c) Protocyon com 873.746 faces e (d) Protocyon com 43.686 faces.

3.2.2 Geração das Imagens Pré-Renderizadas

O visualizador 2 utiliza imagens pré-renderizadas dos modelo 3D, as quais são utilizadas como ponto de referência para a visualização. Assim, o usuário pode navegar pelas imagens pré-renderizadas e ao escolher uma vista que queira visualizar mais detalhes, pode requisitar a mesma para o servidor.

Para a geração das imagens foi desenvolvido um *software* que simula uma visualização do usuário utilizando setas do teclado para rotacionar o modelo 3D. Essas operações alteram o eixo X e Y do modelo 3D, mantendo o mesmo nível de *zoom*. O modelo 3D é movimentado com ângulos variando de 15 graus à 45 graus, dependendo do modelo 3D utilizado. Ângulos menores que 15 graus geram mais imagens e consequentemente, mais banda é necessária para a transferência das mesmas. Ângulos maiores que 45 graus já não apresentam a sensação de interatividade com o modelo 3D, como podemos observar na Figura 3.12.

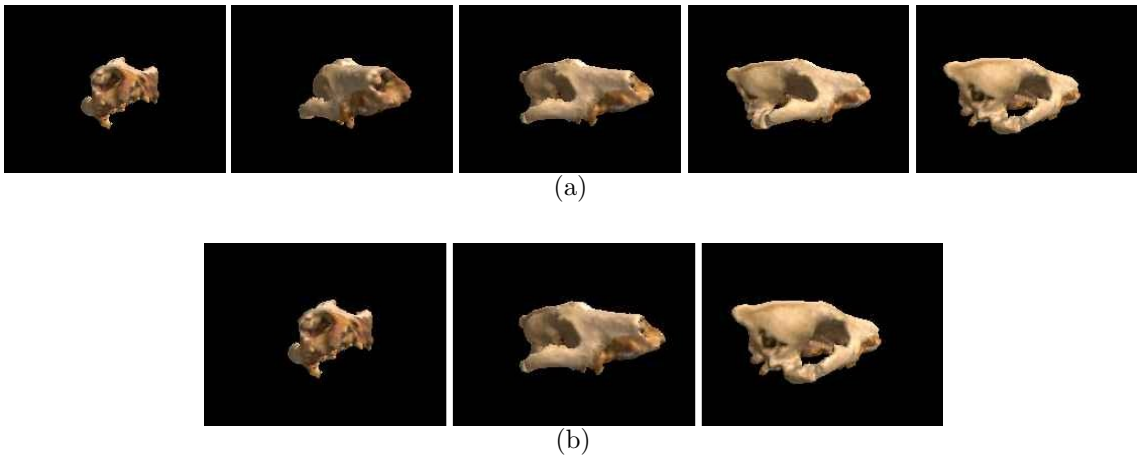


Figura 3.12: Exemplos de imagens pré-renderizadas com ângulos diferentes: (a) Imagens com variação de 30 graus, (b) Imagens com variação de 60 graus.

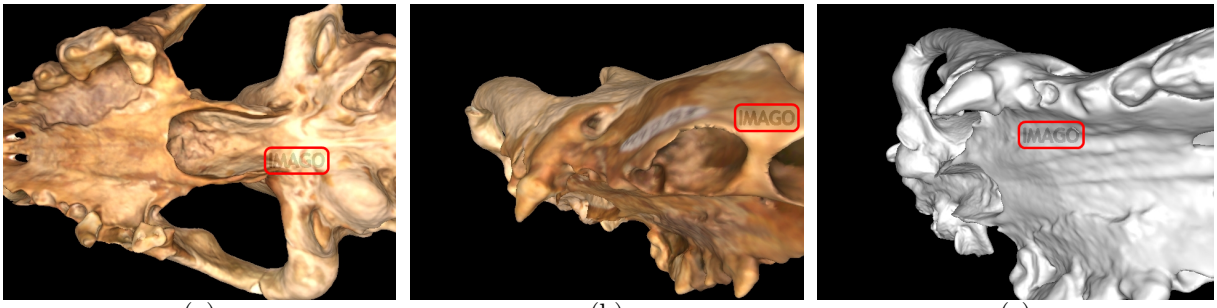
O modelo 3D é renderizado utilizando o ângulo escolhido e as coordenadas necessárias para a geração das imagens são armazenadas. As imagens são geradas utilizando a biblioteca libjpeg[3], com qualidade de 50% e resolução de 220 X 165 *pixels*, resultando na média de 2Kb para cada imagem. Por exemplo, caso um modelo 3D seja pré-renderizado utilizando ângulo de 30 graus, 144 imagens serão geradas, resultando em menos de 300Kb

[3]http://www.ijg.org

a serem transferidos para o usuário. As coordenadas armazenadas são posteriormente carregadas pelo servidor. Assim, quando um cliente envia uma requisição do visualizador 2, o servidor já tem prévio conhecimento de quais coordenadas deve utilizar para a renderização, agilizando o processo de envio da resposta.

3.2.3 Logotipo

No envio de imagens para qualquer visualizador do sistema, um logotipo pode ser adicionado aleatoriamente na imagem, como mostra a Figura 3.13, não interferindo na visualização do modelo 3D. A partir de uma imagem em tons de cinza no formato Joint Photographic Experts Group (JPEG), um módulo do servidor gera uma máscara que contém a largura e altura da imagem, assim como os valores do RGB, de 0 a 255. Essa máscara do logotipo é adicionada, em tempo real, no *buffer* resultante da renderização do modelo 3D (detalhes sobre o *buffer* de renderização na Seção 3.4).



(a) (b) (c)

Figura 3.13: Exemplos de modelos 3D com logotipo do Grupo de Pesquisa IMAGO dispostos aleatoriamente.

No exemplo utilizado, o logotipo inserido tem resolução de 100 x 30 *pixels*, pois o intuito não é atrapalhar a visualização da imagem por parte dos usuários, mas sim garantir que qualquer conteúdo gerado pelo projeto tenha a marca do grupo de pesquisa IMAGO. Futuramente, o logotipo pode ser melhorado, e para aumentar a segurança do conteúdo, marcas d'água podem ser inseridas nas imagens.

3.3 Servidor de Requisições

Ambientes de rede que utilizam *firewall* podem bloquear conexões para determinadas portas como medida de segurança. Por esse motivo, foi criado um servidor para atender as requisições dos clientes, o qual recebe e envia dados através do protocolo HTTPS. Como as portas 80 (HTTP) e 43 (HTTPS) são comumente padrões, garantimos a comunicação entre as aplicações sem a intervenção dos administradores de rede.

O servidor de requisições é responsável por intermediar a comunicação entre o cliente e o servidor de renderização. Esse servidor consiste em uma aplicação Java Servlet, a qual é hospedada no *container* de aplicações Java *web* Apache Tomcat. O ciclo de vida de uma *servlet* é gerenciado pelo *container*, o qual é responsável pelo seu carregamento, instalação, inicialização e recolhimento pelo coletor de lixo. As *servlets* são carregadas em memória apenas na primeira requisição, enquanto que para as demais é criado um outro fluxo de programa. Sendo assim, o acesso concorrente é controlado pelo próprio *container*, sem a necessidade de ser implementado separadamente.

Para requisições dos visualizadores 1 e 2, as mesmas são repassadas para o servidor de renderização, o qual envia imagens de um modelo 3D de alta resolução renderizado. O servidor de renderização (implementado na linguagem C++) envia os dados no formato *little-endian*, enquanto a *servlet* (linguagem Java) os espera no formato de rede *big-endian*. Por esse motivo, é necessário converter os *bytes* antes de enviá-los para o cliente.

O servidor de requisições também é responsável por prever as futuras imagens que poderão ser solicitadas. Conhecendo o comportamento das requisições ao longo da visualização, é possível se antecipar às ações dos usuário. Para o visualizador 3, o servidor de requisições primeiro verifica no *cache* preditivo (detalhes na Seção 3.3.1) se alguma imagem pode ser utilizada para essa requisição. Caso positivo, essa imagem é enviada para o usuário sem o servidor de renderização ser acessado. Caso contrário, a requisição é repassada para o servidor de renderização, o qual envia a imagem.

3.3.1 Cache Preditivo

Alguns eventuais problemas podem acontecer na visualização com renderização remota, como grande latência de rede e a sobrecarga do servidor. A interação do usuário com o modelo 3D pode ser afetada, já que o usuário pode ficar esperando uma imagem por um tempo maior que o desejado.

Prever as futuras vistas que serão visualizadas pelo usuário pode minimizar esses problemas. Com vistas previamente renderizadas, diminui-se a carga de trabalho do servidor para uma determinada requisição. A seguir detalhes sobre a solução utilizada para o *cache* preditivo no sistema de renderização remota desenvolvido.

3.3.1.1 Solução Utilizada

A predição apresenta uma grande vantagem para a visualização do usuário, pois quando uma imagem que já foi renderizada e armazenada pode ser utilizada, evita-se o acesso ao servidor de renderização. Com isso, a espera pela imagem é nitidamente menor, aumentando a interatividade. Por esse motivo, um módulo de *cache* preditivo foi anexado ao servidor de requisições.

Em um ambiente em que o usuário pode visualizar qualquer ângulo do modelo 3D, como no visualizador 1, inúmeras imagens podem ser geradas por predição e nunca serem utilizadas. A movimentação pode ser realizada através do *mouse*, de forma aleatória, podendo não existir um padrão de sequência das ações do usuário. No visualizador 2, o usuário visualiza imagens pré-renderizadas com posições definidas. Assim, torna-se viável a utilização do *cache* preditivo no visualizador 3, pois apesar do usuário ter grande liberdade para visualizar o modelo 3D, a movimentação é realizada apenas através de botões na *interface*.

Cada operação sobre o modelo 3D é realizada por um determinado botão, e as operações mantém um padrão de deslocamento, como por exemplo, seguidas rotações na mesma direção utilizarão valores muito semelhantes de deslocamento. A solução utili-

zada para predição de imagens considera que o usuário vai repetir a última operação de movimentação do modelo 3D realizada, ou seja, rotação, translação ou escala, mantendo a mesma direção. Por exemplo, caso o usuário rotacione o modelo 3D para a direita, a predição gera a próxima imagem realizando rotação a direita a partir do estado atual do modelo 3D.

O servidor de requisições (Seção 3.3) armazena todas as informações necessárias sobre o estado atual do modelo 3D de cada usuário. Ao receber uma requisição, o servidor de requisições aciona o módulo de *cache* preditivo, o qual é executado em paralelo. O *cache* preditivo prediz a próxima imagem a ser visualizada pelo usuário, realizando uma simulação de movimentação através de operações com matriz, utilizando os mesmos parâmetros atuais de textura, iluminação, *wireframe*, velocidade de visualização e qualidade da imagem. Caso a imagem a ser gerada não esteja no banco de dados, a mesma é requisitada para o servidor de renderização e depois é armazenada no banco de dados. O banco de dados possui uma tabela com os seguintes campos e respectivos tipos:

- Nome do modelo 3D (*character varying*(20));
- Largura e altura (*integer*);
- Coordenadas X, Y, Z (*double precision*);
- *Wireframe* (*smallint*);
- Iluminação (*integer*);
- Textura (*smallint*);
- Qualidade da imagem (*integer*);
- Imagem em Base64 (*text*);
- Número de vezes que a imagem foi utilizada (*integer*);

A imagem recebida do servidor de renderização é transformada para Base64 e armazenada no banco de dados, agilizando o processo de recuperação e envio da imagem para o usuário. Em média, uma imagem demora 6 milisegundos para ser gravada no banco de dados. Para recuperar uma imagem, a quantidade de entradas na tabela que armazena as informações deve ser levada em conta. Testes realizados quanto aos tempos de acesso ao conteúdo do *cache* preditivo em relação a quantidade de entradas são apresentados na Figura 3.14.

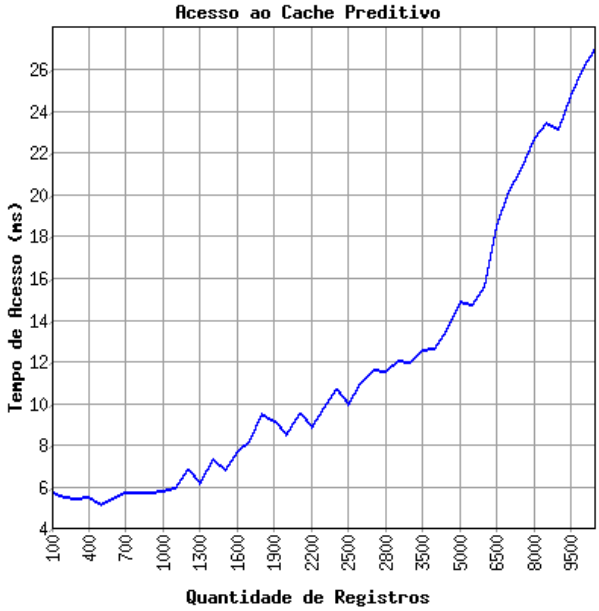


Figura 3.14: Tempo de acesso ao *cache* preditivo, em milisegundos, de acordo com o número de registros no banco de dados.

Observa-se que, geralmente, quanto mais entradas a tabela possui, maior o tempo de acesso e recuperação de uma imagem. A quantidade de dados utilizada para representar a imagem também influencia no tempo de recuperação da mesma. Dessa forma, é necessário manter uma quantidade adequada de entradas para não inviabilizar o uso do *cache* preditivo. O tempo de acesso e recuperação de uma imagem não pode ultrapassar o tempo médio de resposta do servidor de renderização.

A idéia é remover os registros de requisições pouco utilizados, através do uso do campo que indica quantas vezes cada imagem foi recuperada e enviada ao usuário. Para isso, uma função é chamada para cada comando de inserção de registro, a qual verifica se o número máximo de registros já foi alcançado. Caso positivo, é feita a média da quantidade

imagens que foram utilizadas, e todos os registros que foram utilizados inferiormente à média são removidos.

O Gerenciador de Banco de Dados utilizado no sistema é o PostgreSQL[4]. Para a remoção dos registros foi utilizado um recurso de programação chamado de Trigger[5], o qual pode ser visto no Código 3.3.1.

```
1  CREATE LANGUAGE plpgsql;
2
3  CREATE TRIGGER  "quant_req" BEFORE INSERT
4  ON tb_cache_preditivo FOR EACH ROW
5      EXECUTE PROCEDURE delete_req();
6
7
8  CREATE OR REPLACE FUNCTION delete_req()
9  RETURNS trigger LANGUAGE plpgsql
10 AS
11 'declare
12     quant integer := 0;
13     media real := 0;
14 begin
15     quant := (select count(*) from tb_cache_preditivo);
16     if (quant > MAX)
17     then
18         media := (select avg(vezes_utilizada) from tb_cache_preditivo);
19         delete from tb_cache_preditivo where vezes_utilizada <= media;
20     end if;
21     return new;
22 end; ';
```

Código 3.3.1: *Bloco de comandos para criação de um Trigger que é executado antes do comando de inserção na tabela relacionada ao cache preditivo.*

3.4 Servidor de Renderização

O servidor de renderização é o responsável por receber as requisições do servidor de requisições, renderizar o modelo 3D nos parâmetros recebidos e devolver uma imagem no formato JPEG. A Figura 3.15 ilustra o fluxo do servidor, que primeiramente realiza algumas etapas de inicialização, como por exemplo o carregamento dos modelos 3D na memória. Ao receber uma requisição, a mesma é tratada de acordo com o visualizador utilizado e, após a renderização do modelo 3D, uma imagem é enviada.

[4]http://www.postgresql.org

[5]Trigger é um código que pode ser associado a uma ou mais tabelas no banco de dados, executando uma determinada função em situações específicas.

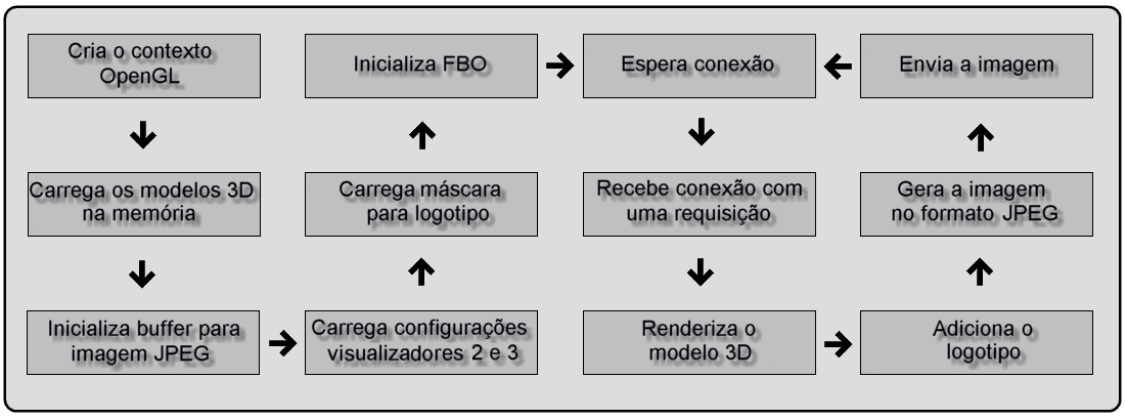


Figura 3.15: Fluxo de inicialização e funcionamento do servidor de renderização remota.

Detalhes sobre a inicialização do servidor de renderização podem ser vistos na Seção 3.4.1. O funcionamento do servidor, assim como a renderização dos modelos 3D e a geração das imagens JPEG, podem ser vistos nas seções seguintes.

3.4.1 Inicialização

Para o servidor de renderização ficar apto a atender às requisições dos clientes, algumas inicializações são realizadas. As operações realizadas na inicialização do servidor de renderização remota são:

- Criação do contexto OpenGL;
- Carregamento dos modelos 3D na memória;
- Inicialização do *buffer* para armazenar a imagem JPEG;
- Carregamento das configurações para os visualizadores 2 e 3;
- Carregamento da máscara do logotipo;
- Inicialização da conexão para atender requisições.

O servidor de renderização não necessita de uma janela visível para realizar a renderização dos modelos 3D. Assim, o contexto gráfico é criado utilizando a biblioteca Xlib,

como mostra a Seção 3.4.1.1. Após a criação do contexto gráfico, os modelos 3D podem ser colocados na memória, sendo a etapa mais demorada da inicialização.

O *buffer* que será utilizado para armazenar a imagem JPEG é inicializado com um tamanho e qualidade definidos, mas que podem ser alterados no decorrer do programa. As configurações relacionadas aos visualizadores 2 e 3, como as imagens pré-renderizadas e as informações geradas na inicialização do sistema, também são carregadas nessa etapa. Finalmente, a máscara do logotipo é carregada (mais informações na Seção 3.2.3) e o servidor espera por requisições.

3.4.1.1 Contexto Gráfico

A Application Programming Interface (API) gráfica OpenGL foi utilizada para a renderização no servidor, assim como utilizado no Plugin IMAGO e 3D IMAGO Viewer [29]. A API disponibiliza comandos que permitem a criação de aplicações 3D interativas, tornando-se padrão na indústria de desenvolvimento de aplicações. A facilidade de aprendizado, a grande quantidade de documentação que pode ser encontrada e a estabilidade da API contribuíram para o fato do OpenGL se tornar um padrão.

O OpenGL fornece acesso a vários recursos do *hardware* gráfico, pois as suas funcionalidades foram implementadas independente da *interface* gráfica utilizada. Uma aplicação em OpenGL pode ser executada em múltiplas plataformas sem alteração no código [26]. A biblioteca OpenGL não oferece as funcionalidades de gerenciamento de janelas e manipulação de eventos, necessitando ser incorporada a qualquer sistema de janelas [5].

Sendo assim, para a utilização do OpenGL é necessário a criação de um contexto gráfico. Um contexto gráfico armazena informações relacionadas à forma como as primitivas serão desenhadas, como por exemplo pontos, linhas, polígonos, a cor a ser utilizada entre outros [33]. Algumas bibliotecas podem ser utilizadas para a criação do contexto gráfico, como o OpenGL Utility Toolkit (GLUT) e GTK. Estas bibliotecas possuem um conjunto de ferramentas que facilitam a construção de programas utilizando o OpenGL, mas criam uma janela visível para inicialização do contexto gráfico.

Finalmente, é necessário uma biblioteca que apenas inicialize o *framebuffer*[6] e forneça uma *interface* para renderização. Foi utilizado a Xlib[7] para a criação do contexto gráfico do servidor de renderização, sem a criação de uma janela visível. Com isso é possível, por exemplo, utilizar um servidor central que acessa vários outros computadores que rodam um servidor de renderização remota, mesmo que esses computadores estejam sendo usados para outros fins.

3.4.2 Funcionamento

Com as inicializações realizadas, o servidor está pronto para atender as requisições dos clientes. Para cada tipo de requisição recebida, uma operação é realizada. Basicamente a seguinte ordem é seguida:

- Definição dos novos parâmetros OpenGL de acordo com a requisição;
- Inicialização do Frame Buffer Object;
- Realização da renderização do modelo 3D;
- Adição do logotipo;
- Geração da imagem JPEG;
- Envio da imagem JPEG.

Dada uma requisição, são definidos os parâmetros OpenGL para a renderização do modelo 3D. A perspectiva da matriz de projeção do OpenGL é definida com os valores de altura e largura recebidos, assim como o *buffer* utilizado para a imagem JPEG. Como as requisições recebidas podem ter parâmetros diferentes, o servidor separa cada parâmetro e o coloca no formato padrão para a renderização do modelo 3D.

[6] *Framebuffer* é um conjunto de *buffers* lógicos (como *buffer* de cor e *buffer* de profundidade) que definem onde a renderização do OpenGL será feita.

[7] Xlib é uma biblioteca que contém um conjunto de funções que são utilizadas como uma *interface* de programação de baixo nível para o X Window System.

O *buffer* utilizado para a renderização do modelo 3D é o Frame Buffer Object (FBO) (detalhes sobre a implementação do FBO podem ser vistos na Seção 3.4.3.1). Este *buffer* é definido com os valores de altura e largura recebidos, e a renderização do modelo 3D é realizada utilizando os demais parâmetros da requisição, como coordenadas, textura, iluminação e *zoom*. O logotipo é adicionado ao *buffer* do FBO e a imagem no formato JPEG é criada e enviada para o servidor de requisições, o qual enviará a mesma para o cliente.

A seguir, mais detalhes sobre como é feita a renderização dos modelos 3D, assim como a utilização do FBO. Maiores informações sobre a criação das imagens JPEG e a inserção do logotipo também são apresentados.

3.4.3 Renderização dos Modelos 3D

Para a renderização dos modelos 3D, é utilizado o OpenGL e a extensão Vertex Buffer Object (VBO). Em geometrias estáticas (ou seja, sem animação), utilizar o VBO proporciona um ganho de desempenho considerável, já que com ele é possível enviar os dados uma única vez para a memória da placa de vídeo. Assim, os dados são gravados na placa de vídeo e altera-se apenas a posição da câmera quando ocorre a mudança na visualização do objeto.

Ao receber uma nova requisição, o servidor realiza pequenas alterações nos parâmetros X, Y e Z do modelo 3D. Essas mudanças afetam a matriz de rotação, pois a mesma utiliza estes parâmetros, influenciando no resultado da renderização do modelo 3D. Com isso, mesmo que o servidor receba duas requisições idênticas, as imagens resultantes da renderização do modelo 3D serão diferentes. Assim, pretendemos evitar uma possível reconstrução do modelo 3D a partir das imagens, já que pequenos desvios na orientação na renderização dos modelos 3D, como mostra a Figura 3.16, alteram a localização exata do modelo 3D na tela, dificultando a junção de várias imagens [19].

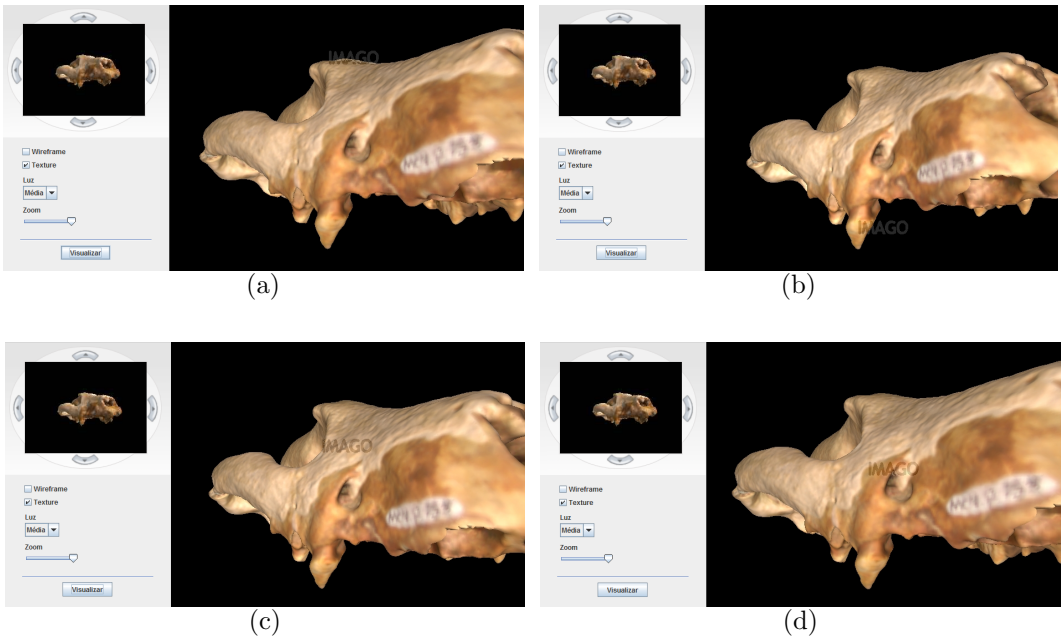


Figura 3.16: Exemplos de imagens geradas a partir de uma mesma requisição. A disposição do modelo 3D na tela é diferente em todas as imagens.

3.4.3.1 Renderização *offscreen* utilizando FBO

Para melhorar o desempenho na renderização, podemos utilizar o FBO, técnica de programação usada para designar um destino diferente da renderização do que o *framebuffer* do OpenGL. Sem o uso do FBO, é realizado a cópia de dados da renderização do *framebuffer* para uma textura. Com a utilização do FBO, as informações como cor ou profundidade são armazenadas diretamente na textura inicialmente definida, e assim, a renderização é realizada utilizando um conjunto menor de comandos, resultando em menos uso de memória e ganho no processamento [12].

Um objeto FBO necessita ter algum outro objeto anexado, como por exemplo um objeto de textura ou outro objeto do OpenGL, chamado de *renderbuffer*[8]. O *renderbuffer* encapsula uma única imagem 2D como uma textura, armazenando os *pixels* resultantes da renderização, e podendo ser compartilhado em múltiplos contextos [12, 17]. Assim, o FBO tem vantagem em relação à utilização de P-Buffers, pois cada P-Buffer usualmente tem seu próprio contexto OpenGL e a troca de contextos entre os P-Buffers é custosa [17, 22].

[8]O *renderbuffer* é um objeto OpenGL que armazena imagens, utilizado para apoiar a renderização *offscreen*.

Antes de inicializar o FBO, devemos criar o objeto de textura, com o uso dos comandos do Código 3.4.1. Primeiro é gerado o nome da textura (linha 1), vinculado a textura ao objeto (linha 2), especificado os parâmetros da textura como filtros e repetições (linhas 3-6) e por fim, a textura é construída (linhas 7-8). Para a criação do objeto FBO, é gerado o nome do objeto (linha 10), associado a textura 2D criada ao objeto FBO (linha 11) e anexado a textura ao objeto FBO (linhas 12-13).

```
1  glGenTextures(1, &textureId);
2  glBindTexture(GL_TEXTURE_2D, textureId);
3  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
4  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
5  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
6  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
7  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, WIDTH, HEIGHT, 0, GL_BGRA,
8          GL_UNSIGNED_BYTE, NULL);
9
10 glGenFramebuffersEXT(1, &fbId);
11 glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbId);
12 glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
13         GL_TEXTURE_2D, textureId, 0);
```

Código 3.4.1: *Criação do objeto de textura, criação do objeto FBO e vinculação da textura com o FBO.*

Com o objeto FBO criado, deve-se gerar o objeto de *renderbuffer*. Como mostra o código 3.4.2, é criado um *buffer* de profundidade e anexado ao *renderbuffer* (linhas 1-2). Após a alocação da memória (linhas 3-4), é anexado o *renderbuffer* ao FBO (linhas 5-6). Para checar se o FBO foi corretamente criado, utiliza-se os comandos das linha 7 e 8.

```
1  glGenRenderbuffersEXT(1, &depth_rb);
2  glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depth_rb);
3  glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT32,
4          WIDTH, HEIGHT);
5  glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT,
6          GL_RENDERBUFFER_EXT, depth_rb);
7  if (glCheckFramebufferStatusEXT(GL_FRAMEBUFFER_EXT) !=GL_FRAMEBUFFER_COMPLETE_EXT)
8          glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
```

Código 3.4.2: *Criação do objeto renderbuffer e seu vínculo com o objeto FBO.*

Para a renderização utiliza-se o Código 3.4.3. Limpa-se o *buffer* (linha 1), desabilita a transparência (linha 2), habilita o *z-buffer* (linha 3), realiza a renderização (linha 4) e obtém os resultados da renderização (linhas 5-6). Para apagar os *buffers* e a textura,

utiliza-se as linhas 8-10. Podemos voltar a realizar a renderização utilizando o *framebuffer* do OpenGL (linha 11), voltando aos estados anteriores do OpenGL (linhas 12-13).

```
1  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
2  glDisable(GL_BLEND);
3  glEnable(GL_DEPTH_TEST);
4  render();
5  glReadBuffer(GL_DEPTH_ATTACHMENT_EXT);
6  glReadPixels(0, 0, param.width, param.height, GL_RGB, GL_UNSIGNED_BYTE, RESULT);
7
8  glDeleteTextures(1, &textureId);
9  glDeleteRenderbuffersEXT(1, &depth_rb);
10 glDeleteFramebuffersEXT(1, &fbId);
11 glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
12 glEnable(GL_BLEND);
13 glDisable(GL_DEPTH_TEST);
```

Código 3.4.3: *Realiza-se a renderização do modelo 3D com FBO e obtém o resultado.*

3.4.4 Adição do Logotipo

Com a renderização concluída, o logotipo pode ser adicionado aos dados contidos no *buffer* do FBO. Como o logotipo pode coincidir com alguma parte do modelo 3D, são encontrados o conjunto de *pixels* pertencentes a região que contém o modelo 3D na imagem. Entre estes *pixels*, um *pixel* de referência é escolhido aleatoriamente para iniciar a adição do logotipo.

O logotipo é adicionado a partir do *pixel* de referência, seguindo a ordem: da esquerda para a direita, de cima para baixo. É utilizada a máscara de logotipo gerada, explicada na Seção 3.2.3. Altera-se os valores RGB dos *pixels* afetados pela máscara com valores que dão o efeito de transparência do logotipo na imagem.

3.4.5 Geração das Imagens JPEG

Após a renderização do modelo 3D e adição do logotipo, os dados estão armazenados no *buffer* FBO na memória. Na renderização de um cenário com 800 x 600 *pixels* por exemplo, o *buffer* tem tamanho de 800 x 600 x 3 (3 para cada canal de cor RGB), resultando em 1440000 *bytes* armazenados na memória.

Para a criação da imagem, os dados contidos no *buffer* do FBO são transformados em uma imagem no formato JPEG. Para isso, foi utilizada a biblioteca libjpeg, mantida pelo Independent JPEG Group. A libjpeg permite ler e gravar arquivos utilizando o formato JPEG, incluindo também algumas funções adicionais como pré-processamento da imagem antes da compressão JPEG, pós-processamento da imagem depois da descompressão, conversão de cores, quantização da cor entre outros.

Para a compressão das imagens, a libjpeg aceita como entrada uma matriz retangular de *pixels*, a largura e altura da matriz e a qualidade da imagem a ser gerada. Quanto maior a qualidade, maior o tamanho final da imagem JPEG gerada, podendo a qualidade variar de 0% a 100%. Como retorno da compressão temos um *buffer* com uma imagem JPEG e o tamanho dessa imagem em Kbytes. Também utilizamos a libjpeg no Plugin IMAGO para realizar a descompressão da imagem recebida, podendo assim ser renderizada na tela com comandos OpenGL.

CAPÍTULO 4

CASO DE ESTUDO: MUSEU VIRTUAL 3D DO IMAGO

O projeto do Museu Virtual 3D do Grupo IMAGO possui um processo de preservação digital que envolve desde a aquisição de dados sobre os objetos até a disponibilização dos modelos 3D através da Internet. Neste projeto, foram preservados diferentes tipos de patrimônios pertencentes a acervos culturais e naturais, tais como: Museu de Belas Artes da UFMG (Figura 4.1a), Acervo da Reitoria (Figura 4.1b), Coleções Biológicas da UFPR (Figura 4.1c), Museu de Arqueologia e Etnologia da UFPR (Figura 4.1d), Museu Metropolitano de Arte Curitiba (Figura 4.1e) e Museu de Ciências Naturais da UFPR.

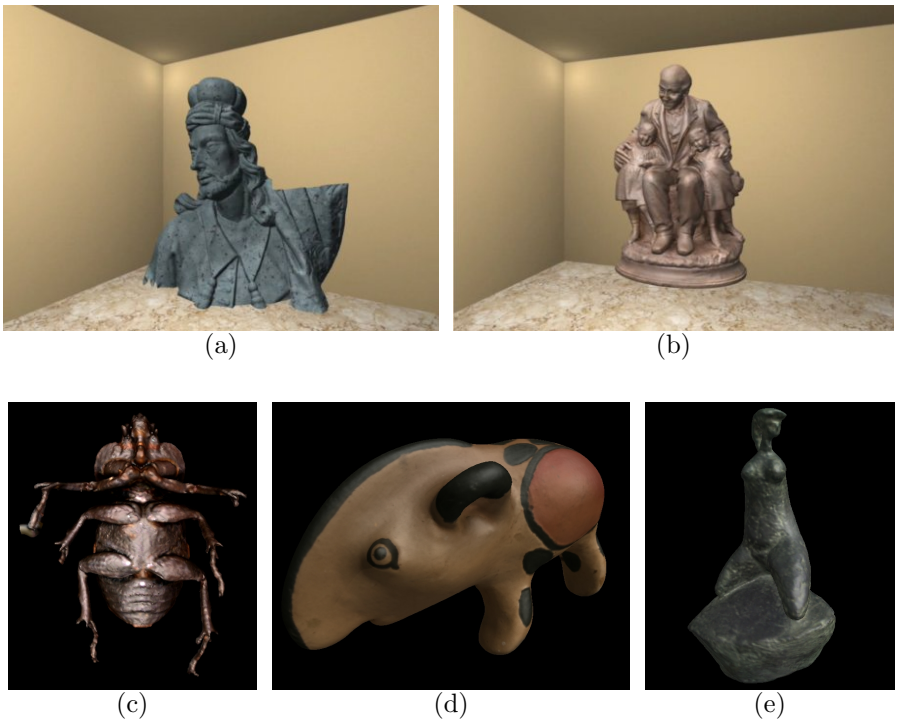
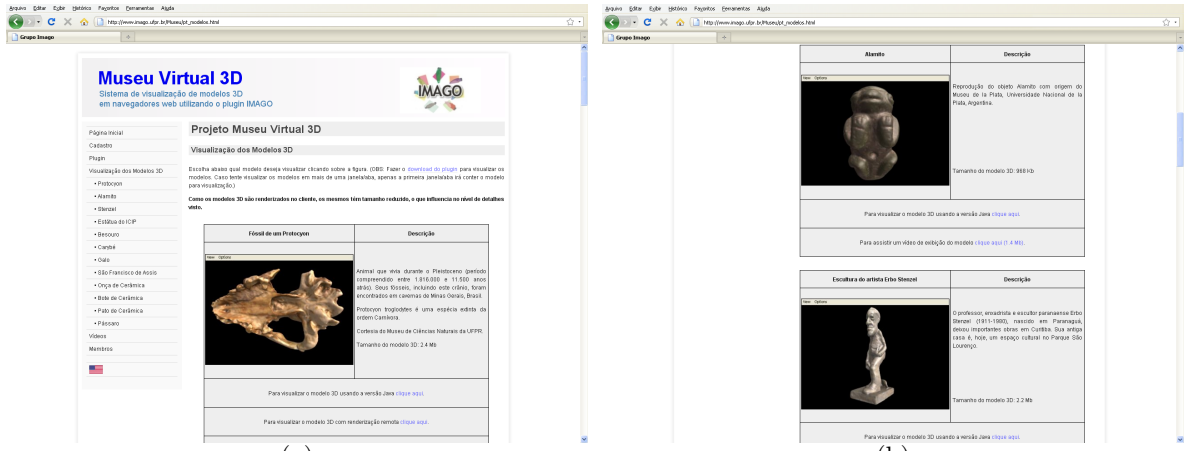


Figura 4.1: Exemplos de modelos 3D preservados: (a) Profeta Habacuc, feito pelo artista Aleijadinho, (b) Estátua presente no gabinete do reitor da UFPR, (c) Besouro, (d) Anta de cerâmica manufaturada por índios da tribo Wauja, (e) Obra do artista Carybé.

Os modelos 3D utilizados nesse sistema foram gerados utilizando o *pipeline* de reconstrução 3D desenvolvido para a preservação digital de patrimônios naturais e culturais [36, 37]. Primeiramente são obtidas imagens coloridas e imagens de profundidade

através de um *scanner* a *laser* de alta resolução (Minolta Vivid 910), e com uma câmera fotográfica profissional (Canon EOS5D) são capturadas imagens de alta resolução para a geração de texturas. Depois, as vistas (obtidas através das imagens de profundidade) são alinhadas e integradas para obter uma malha que corresponde ao modelo 3D do objeto. Finalmente, uma textura de alta qualidade é aplicada ao modelo 3D reconstruído [2, 3].

O Museu Virtual 3D[1] está disponível na Internet, sendo necessário efetuar um cadastro e posteriormente um *login* para acesso ao conteúdo. A página *web* do Museu Virtual 3D (Figura 4.2) apresenta uma breve descrição sobre os objetos e também disponibiliza vídeos dos modelos 3D renderizados. A visualização é feita no navegador *web*, através da utilização do Plugin IMAGO ou do IMAGO 3D Viewer [29].



(a) (b)

Figura 4.2: Páginas *web* do Museu Virtual 3D.

4.1 Sistema de Renderização Remota e a Disponibilização dos Modelos 3D

A primeira forma de disponibilização dos modelos 3D adotada pelo projeto foi a utilização de um Sistema para Visualização de Museu Virtual 3D descrito em [30]. O sistema possui a arquitetura cliente-servidor, em que modelos 3D são armazenados em um banco de dados no servidor, os quais são enviados para o cliente para ser feita a visualização. O servidor também é responsável pelo cadastramento e autenticação do usuário no sistema.

[1] http://www.imago.ufpr.br/Museu

São disponibilizados modelos 3D de alta e baixa resolução para visualização, sendo a maioria disponível em baixa resolução. Modelos 3D de alta resolução podem não ser renderizados no cliente devido a baixa capacidade computacional do computador do usuário. Essa limitação, e o fato de que algumas obras possuem direitos autorais, por exemplo objetos pertencentes a Fundação Cultural de Curitiba (FCC), são os principais motivos para não disponibilizar os modelos 3D de alta resolução.

Alguns problemas ocorrem devido à disponibilização de modelos 3D com resolução variada. Com modelos 3D de alta resolução, o usuário poderá observar mais detalhes da geometria dos objetos, porém o tempo de *download* dos arquivos de modelos 3D é consideravelmente maior se comparado aos modelos 3D de baixa resolução.

A qualidade da visualização pode ser potencializada ou prejudicada dependendo do computador do usuário. Modelos 3D com alta resolução demandam espaço de memória para serem armazenados e a qualidade da renderização e a taxa de quadros por segundo são dependentes de algumas características do computador. Placas de vídeo sem aceleração 3D não são capazes de renderizar os modelos 3D, não permitindo ao usuário acessar o conteúdo 3D disponibilizado.

O Sistema de Renderização Remota desenvolvido contribuiu para amenizar esses problemas. A utilização de imagens dos modelos 3D de alta resolução permite ao usuário observar mais detalhes dos objetos comparado apenas à interação com modelos 3D de baixa resolução. Os visualizadores desenvolvidos também agregaram acessibilidade ao sistema, já que é disponibilizado uma abordagem para cada tipo de usuário, considerando os recursos presentes em seus computadores.

4.2 Testes Realizados

Com o sistema de renderização remota desenvolvido e inserido no Museu Virtual 3D do IMAGO, alguns testes foram realizados para avaliar o visualizador 1 e o servidor. Foram utilizados 4 modelos 3D disponíveis no Museu Virtual 3D: Protocyon, Alamito, Stenzel e Carybe. Para cada modelo 3D foram realizadas 10 operações idênticas simulando a

visualização do usuário. As Figuras 4.3, 4.4, 4.5 e 4.6 exemplificam o resultado visual das ações, as quais são listadas a seguir:

- **Ação 1:** Modelo 3D na posição inicial;

- **Ação 2:** *Zoom in* - 3 vezes;

- **Ação 3:** Rotação para cima - 10 vezes;

- **Ação 4:** Rotação para direita - 5 vezes;

- **Ação 5:** Rotação para baixo - 6 vezes;

- **Ação 6:** Rotação para esquerda - 4 vezes;

- **Ação 7:** *Zoom in* - 2 vezes;

- **Ação 8:** Remoção da textura;

- **Ação 9:** Posição inicial e opções padrões. *Zoom out* - 3 vezes;

- **Ação 10:** Posição inicial e luz ambiente um nível mais forte.

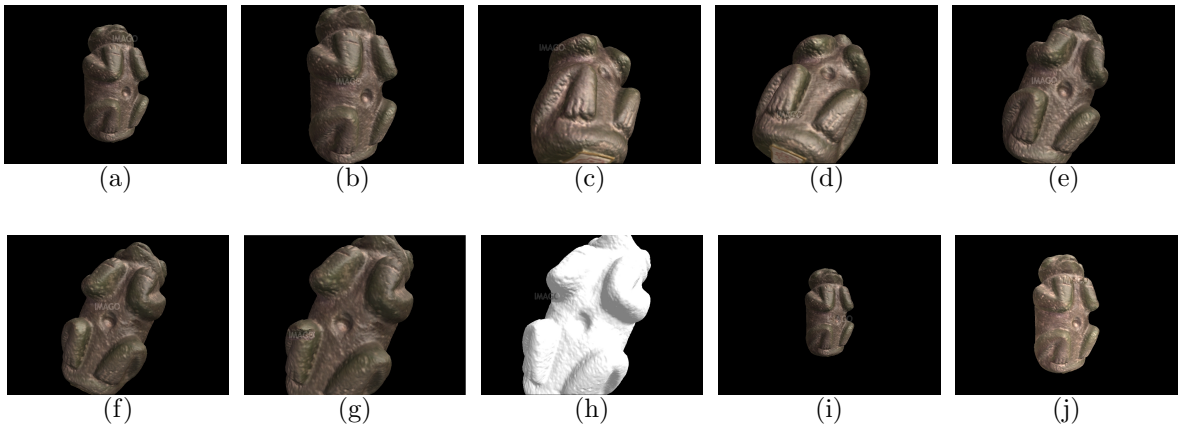


Figura 4.3: Requisições do cliente para o servidor na visualização do modelo 3D Alamito.

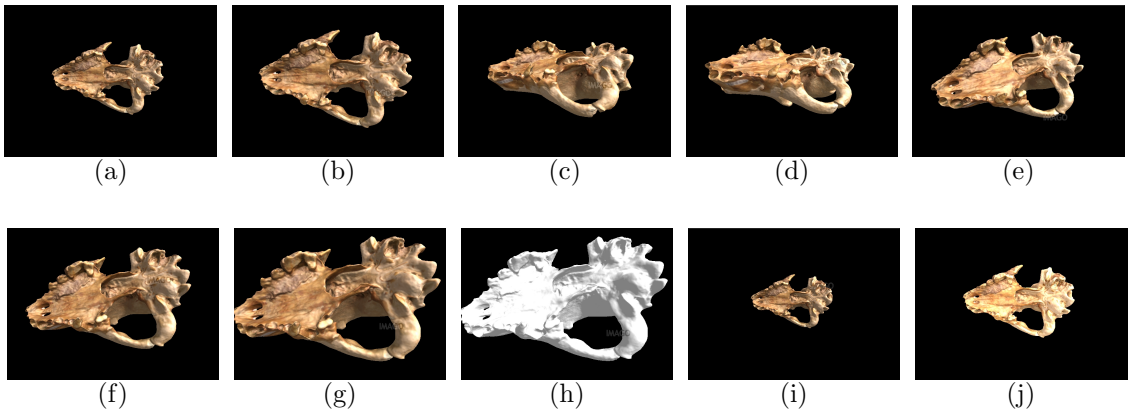


Figura 4.4: Requisições do cliente para o servidor na visualização do modelo 3D Protocyon.

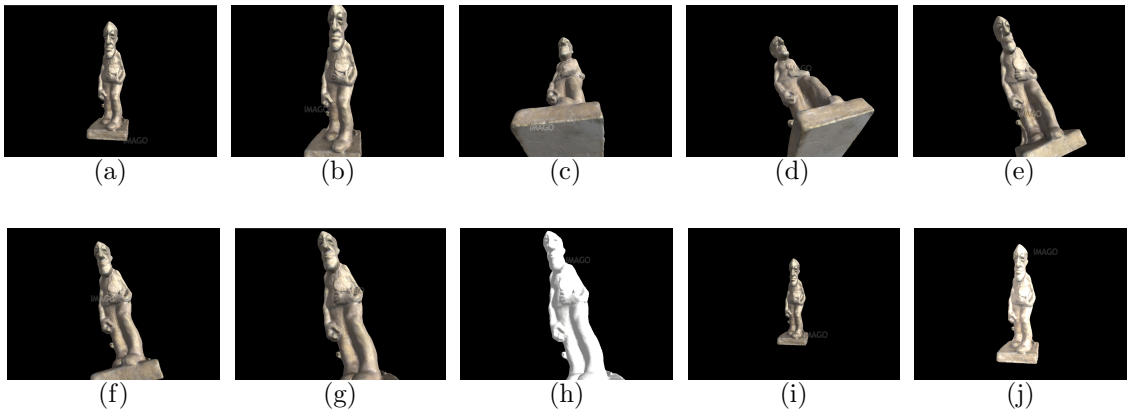


Figura 4.5: Requisições do cliente para o servidor na visualização do modelo 3D do Stenzel.

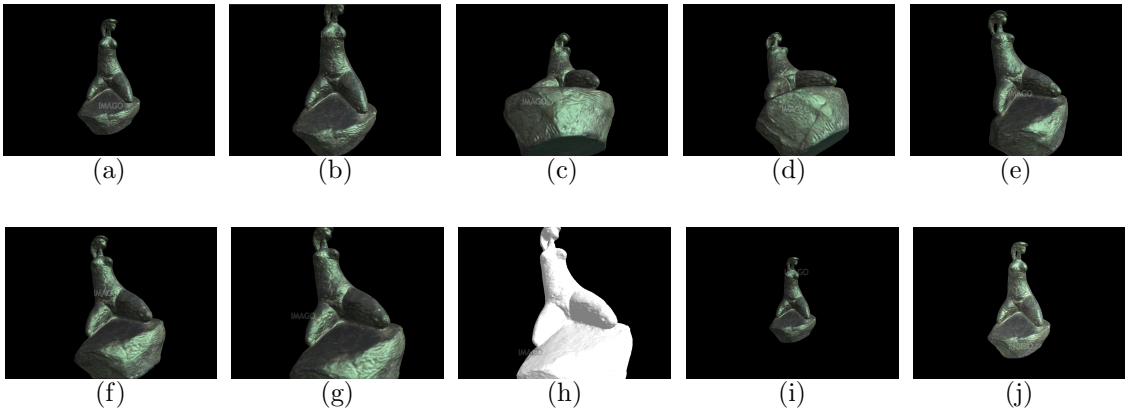


Figura 4.6: Requisições do cliente para o servidor na visualização do modelo 3D do Carybé.

4.2.1 Cliente

Os testes no cliente utilizaram os visualizadores disponíveis do Museu Virtual 3D e um computador com a seguinte configuração: processador Intel Core 2 6300 1.86Ghz, 2GB de memória e placa de vídeo GeForce 6500LE.

A Tabela 4.1 mostra o número de faces e a quantidade de quadros por segundo na renderização dos modelos 3D disponibilizados para a visualização no cliente. Como já dito anteriormente, são enviados modelos 3D de baixa resolução para interação. Os modelos 3D de baixa resolução foram gerados utilizando o método de simplificação visto na Seção 3.2.1.

Tabela 4.1: Informações sobre os modelos 3D de baixa resolução disponibilizados para visualização no cliente.

Modelo 3D	Quantidade de faces	Quadros por segundo
Protocyon	51292 faces	203
Alamito	37456 faces	198
Stenzel	48084 faces	221
Carybé	79664 faces	189

A Tabela 4.2 mostra os testes realizados no cliente utilizando o Plugin IMAGO e o IMAGO 3D Viewer. São exibidos o tempo gasto entre pedir e receber a imagem do servidor para cada ação realizada, assim como o tempo gasto para ler a imagem da memória e desenhá-la na tela.

Nos testes da Tabela 4.2 observa-se que o IMAGO 3D Viewer gasta um tempo relativamente maior para exibir as imagens na tela comparado ao Plugin IMAGO. Para exibir as imagens usamos o comando glDrawPixel() da OpenGL, através da *interface* fornecida pelo JOGL. Um problema que ocorre nessa abordagem é a diferença dos formatos de armazenamento dos dados utilizado pelo Java (*big-endian*) e pelo OpenGL nativo (*little-endian*). Outro problema é que o OpenGL inicia a exibição da imagem na posição (0,0) a partir do canto inferior esquerdo enquanto o Java inicia a partir do canto superior esquerdo.

Para corrigir esses problemas é necessário então converter os dados da imagem para um modelo de cor que o OpenGL entende, usar um *array* de *bytes* para manter os dados no formato adequado, além de inverter a imagem verticalmente para exibí-la corretamente na tela. A realização desses procedimentos são a causa do maior tempo gasto pelo IMAGO 3D Viewer. Mais detalhes sobre como resolver esses problemas podem ser vistos no portal Java Net[2].

[2]http://today.java.net/pub/a/today/2003/09/11/jogl2d.html

Ações do usuário	Tamanho da imagem	Tempo entre pedido e recebimento		Tempo para desenhar a imagem	
		Plugin IMAGO	IMAGO 3D Viewer	Plugin IMAGO	IMAGO 3D Viewer
Ação 1 - Figura 4.4a	81861 bytes	306 ms	535 ms	29 ms	603 ms
Ação 2 - Figura 4.4b	124214 bytes	254 ms	214 ms	33 ms	646 ms
Ação 3 - Figura 4.4c	108086 bytes	132 ms	270 ms	37 ms	237 ms
Ação 4 - Figura 4.4d	111355 bytes	255 ms	155 ms	24 ms	227 ms
Ação 5 - Figura 4.4e	123356 bytes	160 ms	182 ms	28 ms	227 ms
Ação 6 - Figura 4.4f	122232 bytes	156 ms	173 ms	27 ms	238 ms
Ação 7 - Figura 4.4g	164202 bytes	149 ms	199 ms	33 ms	238 ms
Ação 8 - Figura 4.4h	107634 bytes	240 ms	152 ms	27 ms	230 ms
Ação 9 - Figura 4.4i	55592 bytes	128 ms	152 ms	22 ms	224 ms
Ação 10 - Figura 4.4j	88075 bytes	130 ms	159 ms	24 ms	228 ms

(a) Protocyon.

Ações do usuário	Tamanho da imagem	Tempo entre pedido e recebimento		Tempo para desenhar a imagem	
		Plugin IMAGO	IMAGO 3D Viewer	Plugin IMAGO	IMAGO 3D Viewer
Ação 1 - Figura 4.3a	71133 bytes	147 ms	414 ms	32 ms	588 ms
Ação 2 - Figura 4.3b	107627 bytes	116 ms	159 ms	26 ms	237 ms
Ação 3 - Figura 4.3c	105585 bytes	123 ms	160 ms	26 ms	245 ms
Ação 4 - Figura 4.3d	102225 bytes	114 ms	158 ms	29 ms	232 ms
Ação 5 - Figura 4.3e	106118 bytes	114 ms	191 ms	27 ms	234 ms
Ação 6 - Figura 4.3f	105612 bytes	117 ms	160 ms	28 ms	227 ms
Ação 7 - Figura 4.3g	125633 bytes	121 ms	200 ms	26 ms	257 ms
Ação 8 - Figura 4.3h	80929 bytes	104 ms	123 ms	27 ms	239 ms
Ação 9 - Figura 4.3i	48307 bytes	98 ms	142 ms	23 ms	228 ms
Ação 10 - Figura 4.3j	79002 bytes	111 ms	131 ms	25 ms	247 ms

(b) Alamito.

Ações do usuário	Tamanho da imagem	Tempo entre pedido e recebimento		Tempo para desenhar a imagem	
		Plugin IMAGO	IMAGO 3D Viewer	Plugin IMAGO	IMAGO 3D Viewer
Ação 1 - Figura 4.5a	48987 bytes	170 ms	412 ms	27 ms	571 ms
Ação 2 - Figura 4.5b	67319 bytes	114 ms	170 ms	26 ms	231 ms
Ação 3 - Figura 4.5c	71603 bytes	117 ms	167 ms	24 ms	228 ms
Ação 4 - Figura 4.5d	76481 bytes	117 ms	162 ms	27 ms	235 ms
Ação 5 - Figura 4.5e	71654 bytes	117 ms	212 ms	27 ms	222 ms
Ação 6 - Figura 4.5f	63683 bytes	115 ms	155 ms	22 ms	226 ms
Ação 7 - Figura 4.5g	68836 bytes	121 ms	182 ms	23 ms	240 ms
Ação 8 - Figura 4.5h	48252 bytes	103 ms	128 ms	26 ms	242 ms
Ação 9 - Figura 4.5i	35156 bytes	110 ms	133 ms	24 ms	218 ms
Ação 10 - Figura 4.5j	52338 bytes	123 ms	137 ms	30 ms	231 ms

(c) Stenzel.

Ações do usuário	Tamanho da imagem	Tempo entre pedido e recebimento		Tempo para desenhar a imagem	
		Plugin IMAGO	IMAGO 3D Viewer	Plugin IMAGO	IMAGO 3D Viewer
Ação 1 - Figura 4.6a	55172 bytes	150 ms	436 ms	27 ms	616 ms
Ação 2 - Figura 4.6b	82241 bytes	121 ms	176 ms	31 ms	247 ms
Ação 3 - Figura 4.6c	91765 bytes	145 ms	178 ms	29 ms	222 ms
Ação 4 - Figura 4.6d	84765 bytes	224 ms	175 ms	24 ms	249 ms
Ação 5 - Figura 4.6e	84993 bytes	121 ms	186 ms	26 ms	224 ms
Ação 6 - Figura 4.6f	86806 bytes	227 ms	243 ms	28 ms	222 ms
Ação 7 - Figura 4.6g	103771 bytes	134 ms	159 ms	30 ms	238 ms
Ação 8 - Figura 4.6h	68712 bytes	125 ms	141 ms	28 ms	226 ms
Ação 9 - Figura 4.6i	38175 bytes	137 ms	134 ms	23 ms	218 ms
Ação 10 - Figura 4.6j	61505 bytes	113 ms	160 ms	24 ms	234 ms

(d) Carybe.

Tabela 4.2: Testes com a visualização de modelos 3D pelo usuário utilizando Plugin IMAGO e IMAGO 3D Viewer.

4.2.2 Servidor

Durante a inicialização do servidor de renderização, são carregados os modelos 3D de alta resolução na memória. A Tabela 4.3 mostra o número de faces que cada um possui, assim

como o tempo de carregamento em memória. O carregamento das configurações relacionadas ao visualizador 2 e a adição do logotipo no resultado da renderização demoram, juntos, em média 3 milisegundos.

Tabela 4.3: Informações sobre os modelos 3D de alta resolução utilizados no servidor.

Modelo 3D	Quantidade de faces	Tempo de carregamento
Protocyon	873.746 faces	5.374 ms
Alamito	342.520 faces	2.089 ms
Stenzel	666.900 faces	4.157 ms
Carybé	714.696 faces	4.402 ms

Para as mesmas ações do usuário vistos na Seção 4.2.1, temos o tempo total gasto no servidor para renderizar e criar a imagem do modelo 3D de alta resolução, com resolução de 800 x 600 *pixels* e qualidade de imagem 100%. Observa-se na Tabela 4.4 que os tempos para atender as requisições do Plugin IMAGO são semelhantes aos do IMAGO 3D Viewer. A semelhança ocorre porque os parâmetros das requisições são os mesmos, ou seja, a mesma imagem foi requisitada. Para uma mesma requisição, o que pode mudar na resposta do servidor é o tamanho da imagem, já que a incorporação do logotipo pode ser feita em regiões distintas do modelo 3D. O tempo de adição do logotipo demora, em média, 2 milisegundos.

Para os testes de renderização no servidor, dois *drivers* de placa de vídeo foram utilizados. Como a placa de vídeo do servidor de renderização remota é o modelo 8800gt da NVIDIA, o *driver* proprietário da mesma é utilizado por padrão. Um *driver* de implementação livre específico para renderização *offscreen* também foi testado, chamado de OffScreen Mesa (OSMesa).

O OSMesa é um *driver* que realiza a renderização de imagens 3D em memória ao invés de utilizar uma janela na tela. Atualmente faz parte do projeto XFree86/DRI, podendo ser utilizado com o libGL.so e XFree86/DRI. O OSMesa se destaca pela sua conformidade técnica com o OpenGL versão 2.1 e por não necessitar de um gerenciador de janelas para a criação do contexto gráfico. Testes comparativos em relação ao desempenho dos dois *drivers* podem ser vistos na Tabela 4.5.

Requisição	Tempo total do servidor	
	Plugin IMAGO	IMAGO 3D Viewer
Figura 4.4a	184 ms	187 ms
Figura 4.4b	135 ms	125 ms
Figura 4.4c	117 ms	121 ms
Figura 4.4d	124 ms	118 ms
Figura 4.4e	144 ms	128 ms
Figura 4.4f	132 ms	131 ms
Figura 4.4g	122 ms	123 ms
Figura 4.4h	112 ms	115 ms
Figura 4.4i	115 ms	116 ms
Figura 4.4j	112 ms	114 ms

(a) Protocyon.

Requisição	Tempo total do servidor	
	Plugin IMAGO	IMAGO 3D Viewer
Figura 4.3a	130 ms	156 ms
Figura 4.3b	101 ms	111 ms
Figura 4.3c	105 ms	110 ms
Figura 4.3d	99 ms	115 ms
Figura 4.3e	96 ms	130 ms
Figura 4.3f	99 ms	124 ms
Figura 4.3g	103 ms	143 ms
Figura 4.3h	85 ms	85 ms
Figura 4.3i	87 ms	105 ms
Figura 4.3j	99 ms	102 ms

(b) Alamito.

Requisição	Tempo total do servidor	
	Plugin IMAGO	IMAGO 3D Viewer
Figura 4.5a	158 ms	153 ms
Figura 4.5b	102 ms	120 ms
Figura 4.5c	104 ms	116 ms
Figura 4.5d	103 ms	109 ms
Figura 4.5e	103 ms	146 ms
Figura 4.5f	101 ms	107 ms
Figura 4.5g	108 ms	105 ms
Figura 4.5h	93 ms	96 ms
Figura 4.5i	99 ms	102 ms
Figura 4.5j	112 ms	102 ms

(c) Stenzel.

Requisição	Tempo total do servidor	
	Plugin IMAGO	IMAGO 3D Viewer
Figura 4.6a	134 ms	156 ms
Figura 4.6b	108 ms	112 ms
Figura 4.6c	128 ms	124 ms
Figura 4.6d	108 ms	113 ms
Figura 4.6e	106 ms	113 ms
Figura 4.6f	107 ms	149 ms
Figura 4.6g	118 ms	117 ms
Figura 4.6h	113 ms	104 ms
Figura 4.6i	126 ms	103 ms
Figura 4.6j	101 ms	116 ms

(d) Carybe.

Tabela 4.4: Testes do servidor de renderização remota.

O tempo total gasto pelo servidor para renderizar e gerar a imagem requisitada é nitidamente menor utilizando o *driver* da NVIDIA comparado ao *driver* OSMesa. Isso acontece porque o *driver* NVIDIA utiliza renderização por *hardware*, já que possui acesso direito ao mesmo, enquanto o OSMesa não provê aceleração de hardware, simulando as extensões do OpenGL e realizando a renderização por *software*. Assim, o principal gargalo do OSMesa é a renderização do modelo 3D, já que o tempo de renderização utilizando o driver NVIDIA é aproximadamente 0 milisegundos.

O OSMesa leva vantagem nos tempos de utilização do *buffer* FBO, pois demora aproximadamente 0 milisegundos para criação do *buffer* FBO, e apenas 8 milisegundos em média para leitura do resultado. Utilizando o driver NVIDIA, a criação do *buffer* FBO demora aproximadamente 25 milisegundos, enquanto o acesso ao resultado da renderização depende do modelo 3D, como mostra a Tabela 4.5.

Requisição	*Driver* NVIDIA		*Driver* OSmesa	
	Tempo de acesso ao *buffer* FBO	Tempo total do servidor	Tempo de renderização	Tempo total do servidor
Figura 4.4a	81 ms	184 ms	669 ms	728 ms
Figura 4.4b	38 ms	135 ms	834 ms	993 ms
Figura 4.4c	38 ms	117 ms	722 ms	789 ms
Figura 4.4d	38 ms	124 ms	711 ms	776 ms
Figura 4.4e	61 ms	144 ms	712 ms	866 ms
Figura 4.4f	38 ms	132 ms	734 ms	800 ms
Figura 4.4g	35 ms	122 ms	768 ms	841 ms
Figura 4.4h	32 ms	112 ms	635 ms	698 ms
Figura 4.4i	38 ms	115 ms	639 ms	696 ms
Figura 4.4j	37 ms	112 ms	669 ms	728 ms

(a) Protocyon.

Requisição	*Driver* NVIDIA		*Driver* OSmesa	
	Tempo de acesso ao *buffer* FBO	Tempo total do servidor	Tempo de renderização	Tempo total do servidor
Figura 4.3a	21 ms	130 ms	306 ms	365 ms
Figura 4.3b	16 ms	101 ms	325 ms	483 ms
Figura 4.3c	16 ms	105 ms	318 ms	402 ms
Figura 4.3d	19 ms	99 ms	326 ms	393 ms
Figura 4.3e	16 ms	96 ms	322 ms	386 ms
Figura 4.3f	22 ms	99 ms	326 ms	459 ms
Figura 4.3g	19 ms	103 ms	333 ms	475 ms
Figura 4.3h	14 ms	85 ms	282 ms	345 ms
Figura 4.3i	16 ms	87 ms	286 ms	382 ms
Figura 4.3j	23 ms	99 ms	300 ms	359 ms

(b) Alamito.

Requisição	*Driver* NVIDIA		*Driver* OSmesa	
	Tempo de acesso ao *buffer* FBO	Tempo total do servidor	Tempo de renderização	Tempo total do servidor
Figura 4.5a	53 ms	158 ms	503 ms	558 ms
Figura 4.5b	30 ms	102 ms	529 ms	644 ms
Figura 4.5c	31 ms	104 ms	533 ms	628 ms
Figura 4.5d	30 ms	103 ms	545 ms	659 ms
Figura 4.5e	28 ms	103 ms	527 ms	642 ms
Figura 4.5f	30 ms	101 ms	516 ms	630 ms
Figura 4.5g	30 ms	108 ms	496 ms	610 ms
Figura 4.5h	25 ms	93 ms	401 ms	454 ms
Figura 4.5i	30 ms	99 ms	478 ms	592 ms
Figura 4.5j	40 ms	112 ms	499 ms	555 ms

(c) Stenzel.

Requisição	*Driver* NVIDIA		*Driver* OSmesa	
	Tempo de acesso ao *buffer* FBO	Tempo total do servidor	Tempo de renderização	Tempo total do servidor
Figura 4.6a	34 ms	134 ms	544 ms	600 ms
Figura 4.6b	32 ms	108 ms	575 ms	692 ms
Figura 4.6c	40 ms	128 ms	583 ms	660 ms
Figura 4.6d	32 ms	108 ms	577 ms	637 ms
Figura 4.6e	32 ms	106 ms	570 ms	630 ms
Figura 4.6f	32 ms	107 ms	570 ms	684 ms
Figura 4.6g	32 ms	118 ms	565 ms	680 ms
Figura 4.6h	27 ms	113 ms	455 ms	509 ms
Figura 4.6i	42 ms	126 ms	512 ms	626 ms
Figura 4.6j	31 ms	101 ms	542 ms	598 ms

(d) Carybe.

Tabela 4.5: Comparação da renderização no servidor utilizando os *drivers* NVIDIA e OSMesa.

4.2.2.1 Cache Preditivo

A principal intenção em utilizar a abordagem de *cache* preditivo é reduzir a carga do servidor de renderização remota. Quanto mais imagens forem recuperadas do *cache* preditivo, mais rápida é a exibição das imagens no cliente. Para testar a viabilidade da utilização de um *cache* preditivo, 15 usuários utilizaram o sistema e manipularam modelos 3D utilizando o visualizador 3. Os usuários foram instruídos a investigar o objeto com total liberdade, podendo realizar qualquer operação de movimentação no modelo 3D, limitada a 100 operações.

A Tabela 4.6 mostra que a maioria das imagens requisitadas são obtidas do *cache* preditivo ao invés do servidor de renderização. Observou-se nos testes que os usuários tendem a repetir suas operações anteriores, como por exemplo a rotação em um mesmo eixo. Outra observação relevante diz respeito ao nível de experiência do usuário. Usuários iniciantes em visualização 3D apresentaram dificuldades para posicionar o objeto em uma vista desejada e, por esse motivo, fizeram várias operações sem um determinado padrão.

As imagens no *cache* preditivo foram limitadas ao uso apenas por cada usuário, não sendo compartilhadas entre os outros usuários visualizando o mesmo modelo 3D. O compartilhamento das imagens tende a aumentar a taxa de acertos do *cache* preditivo. Sem o compartilhamento, a média de imagens que o *cache* preditivo gerou após 100 movimentações foi de 91.4 imagens, com média de 68.9% de acerto.

Tabela 4.6: Taxa de acerto do *cache* preditivo.

Usuários	Quantidade de imagens no *cache* preditivo	Taxa de acerto do *cache* preditivo
1	73	59%
2	95	62%
3	82	63%
4	96	81%
5	93	58%
6	99	79%
7	98	81%
8	91	82%
9	90	64%
10	97	63%
11	82	56%
12	98	85%
13	82	66%
14	96	71%
15	99	64%

CAPÍTULO 5

CONCLUSÃO

O desenvolvimento de um sistema de renderização remota permite ampliar os estudos na área de visualização 3D. Atualmente os projetos de computação gráfica vêm gerando um grande volume de dados e têm encontrado dificuldades para disponibilizar estes dados na Internet, tanto pelo tamanho dos mesmos quanto pela preocupação com a preservação dos direitos autorais.

A incorporação da renderização remota em sistemas para visualização de modelos 3D torna estes sistemas seguros e confiáveis. Sendo confiável, curadores de museus, artistas, pesquisadores entre outros poderão aumentar a disponibilização de seus objetos na Internet. Consequentemente, um maior número de informações poderá ser encontrado para realização de pesquisas e estudos remotos.

Aplicações como museus virtuais necessitam, além da segurança dos modelos 3D de objetos, tornar seus sistemas mais acessíveis. A renderização remota permite que usuários possam visualizar o conteúdo 3D disponibilizado na Internet, independente da especificação do computador. O fato do usuário interagir com modelos 3D de baixa resolução ou apenas com imagens não influencia na visualização de detalhes dos modelos 3D.

A incorporação do Sistema de Renderização Remota tornou o Museu Virtual 3D do IMAGO mais acessível e seguro, além de contribuir para a inclusão digital. Os visualizadores desenvolvidos provém formas alternativas de visualização para os variados tipos de usuários e seus computadores, enquanto os modelos 3D realistas e preciosos para a preservação digital ficam armazenados no servidor. Dessa maneira, a visualização dos modelos 3D para as atividades de pesquisa pode ser feita com segurança e com bom

nível de interatividade para os usuários. A utilização dos visualizadores desenvolvidos através de um navegador *web* contribui para a acessibilidade do projeto, diferentemente de projetos como Michelangelo Digital e Virtual Inspector que necessitam de instalações adicionais para visualização do conteúdo 3D.

Considerando a possível queda de interatividade na visualização apenas de imagens, foi desenvolvido um módulo de *cache* preditivo. Apesar da abordagem de predição das imagens parecer simples, os testes mostraram que os usuários tendem a repetir suas operações de movimentação a partir de uma determinada vista. A maioria das imagens solicitadas são obtidas diretamente do *cache* preditivo, o qual possui um tempo de recuperação menor comparado ao acesso ao servidor de renderização. Assim, o tempo de exibição das imagens no cliente é menor e a interatividade é compensada pelo fato de não se utilizar um modelo 3D para interação.

Como trabalhos futuros, pretende-se estudar a incorporação do *cache* preditivo também para os demais visualizadores desenvolvidos neste trabalho, assim como implementar novos algoritmos de predição a partir das estatísticas geradas com a utilização da versão atual. Dependendo da complexidade do modelo 3D, o processo de renderização a fim de gerar uma imagem para o sistema de renderização remota pode ter alto custo computacional. Uma maneira de resolver esse problema é realizar a renderização de forma distribuída. Várias abordagens para esse tipo de processamento podem ser estudadas, como por exemplo o uso de processadores com vários núcleos e utilização de *clusters*.

BIBLIOGRAFIA

[1] Tomas Akenine-Moller, Tomas Moller, e Eric Haines. *Real-Time Rendering*. AK Peters Ltd., 2002.

[2] Beatriz T. Andrade. Utilizando fotografias digitais de alta qualidade na geração de textura para modelos 3D: Uma abordagem prática na preservação digital de acervos culturais e naturais. Dissertação de Mestrado, Universidade Federal do Paraná, 2009.

[3] Beatriz T. Andrade, Olga R. P. Bellon, Luciano Silva, e Alexandre Vrubel. Enhancing color texture quality of 3D models for digital preservation of indigenous ceramic artworks. *Proceedings of IEEE International Conference on Computer Vision, Workshop on eHeritage and Digital Art Preservation*, 2009.

[4] Paul Bao e Douglas Gourlay. Low bandwidth remote rendering using 3D image warping. *IEE Conference Publications*, páginas 61–64, 2003.

[5] Alessandro L. Bicho, Luiz Gonzaga da Silveira Jr, Adailton J. A. da Cruz, e Alberto B. Raposo. Programação gráfica 3D com opengl, open inventor e java 3D. *Revista Eletrônica de Iniciação Científica*, 2(1), 2002.

[6] Grigore C. Burdea e Philippe Coiffet. *Virtual Reality Technology, Second Edition with CD-ROM*. Wiley-IEEE, Junho de 2003.

[7] Marco Callieri, Federico Ponchio, Paolo Cignoni, e Roberto Scopigno. Easy access to huge 3D models of works of art. *Eurographics Italian Chapter Conference*, páginas 29–36, Fevereiro de 2006.

[8] Marco Callieri, Federico Ponchio, Paolo Cignoni, e Roberto Scopigno. Virtual inspector: A flexible visualizer for dense 3D scanned models. *IEEE Computer Graphics and Applications*, 28(1):44–54, 2008.

[9] Ransi Nilaksha De Silva, Wei Cheng, Dan Liu, Wei Tsang Ooi, e Shengdong Zhao. Towards characterizing user interaction with progressively transmitted 3D meshes.

Proceedings of the 7th ACM International Conference on Multimedia, páginas 881–884, New York, NY, USA, 2009. ACM.

[10] Klaus Engel, Ove Sommer, Christian Ernst, e Thomas Ertl. Remote 3D visualization using image-streaming techniques. *International Symposium on Intelligent Multimedia and Distance Education*, páginas 91–96, 1999.

[11] Klaus Engel, Ove Sommer, e Thomas Ertl. A framework for interactive hardware accelerated remote 3D-visualization. *Proceedings of Symposium on Visualization (VisSym)*, páginas 167–177, 2000.

[12] OpenGL extension registry. *Framebuffer Object Techinical Paper*. http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt.

[13] Jerrold A. Friesen e Thomas D. Tarman. Remote high-performance visualization and collaboration. *IEEE Computer Graphics and Applications*, 20(4):45–49, 2000.

[14] Richard Fujimoto. Parallel and distributed simulation. *Winter Simulation Conference*, páginas 122–131, 1999.

[15] Michael Garland. *Quadric-based polygonal surface simplification*. Tese de Doutorado, Pittsburgh, PA, USA, 1999. Chair-Heckbert, Paul.

[16] Michael Garland e Paul S. Heckbert. Surface simplification using quadric error metrics. *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, páginas 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[17] Simon Green. The opengl framebuffer object extension. *Game Developers Conference*, 2005.

[18] Yuezhu Huang, Chenglei Yang, Xiangxu Meng, Xiaoting Wang, e Lu Wang. Remote non-photorealistic rendering of 3D models on mobile devices. *Proceedings of the International Symposium on Pervasive Computing and Applications*, páginas 364–369, 2006.

[19] David Koller, Michael Turitzin, Marc Levoy, Marco Tarini, Giuseppe Croccia, Paolo Cignoni, e Roberto Scopigno. Protected interactive 3D graphics via remote rendering. *ACM Transactions on Graphics*, 23(3):695–703, 2004.

[20] Fabrizio Lamberti, Claudio Zunino, Andrea Sanna, Antonino Fiume, e Marco Maniezzo. An accelerated remote graphics architecture for PDAs. *Proceedings of the International Conference on 3D Web Technology*, páginas 55–62, 2003.

[21] Kyong-Ho Lee, Oliver Slattery, Richang Lu, Xiao Tang, e Victor Mccrary. The state of the art and practice in digital preservation. *Journal of Research of the National Institute of Standards and Technology*, 107(1):93–106, Janeiro de 2002.

[22] Aaron Lefohn, Joe M. Kniss, e John D. Owens. Implementing efficient parallel data structures on GPUs. Matt Pharr, editor, *GPU Gems 2*, páginas 521–545. Addison-Wesley, Março de 2005.

[23] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, e Duane Fulk. The Digital Michelangelo Project: 3D Scanning of Large Statues. *Proceedings of ACM SIGGRAPH 2000*, páginas 131–144, 2000.

[24] David P. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, 21(3):24–35, 2001.

[25] Eric J. Luke e Charles D. Hansen. Semotus visum: a flexible remote visualization framework. *Proceedings of the Conference on Visualization*, páginas 61–68, 2002.

[26] Isabel Harb Manssour e Marcelo Cohen. Introdução à computacao gráfica. *Revista de Informática Teórica e Aplicada (RITA)*, 13(2):43–68, 2006.

[27] William R. Mark, Leonard McMillan, e Gary Bishop. Post-rendering 3D warping. *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, páginas 7–16, New York, NY, USA, 1997. ACM.

[28] Leonard McMillan, Jr. *An image-based approach to three-dimensional computer graphics*. Tese de Doutorado, 1997.

[29] Caroline M. Mendes. Visualização 3D interativa aplicada a preservação digital de acervos naturais e culturais. Dissertação de Mestrado, Universidade Federal do Paraná, 2010.

[30] Caroline M. Mendes, Dyego R. Drees, Olga R. P. Bellon, e Luciano Silva. Sistema para visualização de museu virtual 3D. V Workshop of Undergraduated Work, SIBGRAPI, 2007.

[31] Caroline M. Mendes, Dyego R. Drees, Luciano Silva, e Olga R. P. Bellon. Interactive 3D visualization of natural and cultural assets. *ACM Multimedia 2010 Workshop - Electronic Heritage and Digital Art Preservation (eHeritage)*, Outubro de 2010.

[32] Steffen Prohaska, Andrei Hutanu, Ralf Kahler, e Hans-Christian Hege. Interactive exploration of large remote micro-ct scans. *Proceedings of the Conference on Visualization*, páginas 345–352, 2004.

[33] Randi J. Rost. *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional, Janeiro de 2006.

[34] Dieter Schmalstieg. *The Remote Rendering Pipeline - Managing Geometry and Bandwidth in Distributed Virtual Environments*. Tese de Doutorado, 1997.

[35] Simon Stegmaier, Marcelo Magallón, e Thomas Ertl. A generic solution for hardware-accelerated remote visualization. *Proceedings of the Symposium on Data Visualization*, páginas 87–94, Maio de 2002.

[36] Alexandre Vrubel. Pipeline para reconstrução digital de objetos com scanners 3D de triangulação a laser : aplicação na preservação digital de acervos naturais e culturais. Dissertação de Mestrado, Universidade Federal do Paraná, 2008.

[37] Alexandre Vrubel, Olga R.P. Bellon, e Luciano Silva. A 3D reconstruction pipeline for digital preservation. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:2687–2694, 2009.

[38] Ilmi Yoon e Ulrich Neumann. Ibrac: Image-based rendering acceleration and compression. *Eurographics*, 19:321–330, 2000.

[39] Hong Zhou. A survey on ubiquitous graphics. Relatório técnico, Hong Kong University of Science and Technology, 2005.